MASTER IN COMPUTER SCIENCE

How computer science can assist theatrical practitioners?

Fontaine, Vincent

Award date:
2000

Facultés Universitaires Notre-Dame de la Paix
University of Namur, Belgium
Computer Science Department

How Computer Science Can Assist Theatrical Practitioners ?

Vincent Fontaine

Director: Prof. Jacques Berleur S. J.

Academic year 1999–2000

Master thesis presented in order to obtain the title of *Maître en Informatique*

English	Français
(to) Sketch	Esquisser, dessiner à grands traits
Sketching	Dessin à main levée
Specifications	Cahier des charges Description précise, prescriptions
Stage	Scène (en Pratique)
Stage acting	Jeu de scène
Stage design	Décoration théâtrale
Stage designer	Décorateur de théâtre
(Stage) direction	Direction scénique
(Stage) director	Metteur en scène
(to) Stage-manage	Mettre en scène
Stage manager	Régisseur (d'une pièce)
Stage painter	Peintre de décors
(Stage) play	Pièce de théâtre
Stage playing	Jeu de scène
Stagecraft	Technique de la scène
Stagehand	Machiniste
Staging	Mise en scène (en Pratique)
Theatre hall	Salle théâtrale
Theatre manager	Directeur de théâtre
Theatre production	Mise en scène théâtrale
Theatrical art	Art théâtral
Vanishing point	Point de fuite
Walk-on (part)	Rôle : Figuration
Walk-on actors	Acteur : Figuration
Work	Œuvre

French Translator

English	Français
Actor	Acteur
Apron (stage)	Avant-scène
Audience	Spectateurs
Back of the stage	Dernier plan de la scène
Back stage	Arrière-scène
Bottom stage	Plancher de scène
Brainwave	Idée, trait génie, idée lumineuse
Casting	Théâtre : Distribution des rôles, casting Informatique : Conversion explicite de type
(to) Change of scene	Changer de décor
Character	Personnage
Comedian	Comédien
Computer graphics	Infographie
Conductor	Chef d'orchestre
Costume	Costume
Costum(i)er	Costumier
Craft	Manuel, artisanal
Craft document	Description précise, prescriptions de mise en scènes
(to) Direct	Théâtre, Cinéma : Mettre en scène une pièce, un film Théâtre, Cinéma : diriger un film, des acteurs, etc

English	Français
Direction	Théâtre : Mise en scène (en Pratique et en Abstraction) Cinéma, TV, Radio : Réalisation
Director	Théâtre : Metteur en scène Cinéma, TV, Radio : Réalisateur
Drama	Art dramatique Le théâtre
Drama critic	Critique dramatique
Drama person	Personne du drame
Drama work	Œuvre dramatique
Dramatic criticism	Critique dramatique
Dramatis person(ae)	Personnage(s)
Dramatist	Dramaturge
Dress	Costume
Dress rehearsal	Répétition générale
Emotion	Émotion
Essence of the (drama) work	Intentionnalité du texte, de l'œuvre
Feeling	Émotion Sensation
Floor stage	Plancher de la scène
Footlights	Rampe (de lumière) de l'avant-scène
Forestage	Avant-scène
Front of the stage	Premier plan de la scène
Front stage	Avant-scène
Ideas (wo)man	Concepteur
Live theatre	Théâtre de rue
Lighting technician	Éclairagiste
Model	Maquette
Narrative	Récit, narration
(to) Perform	Jouer, représenter une pièce

English	Français
Performance	Théâtre : Représentation Cinéma : Séance Musique : Interprétation
Performer	Théâtre :, Acteur Musique : Exécutant, interprète, artiste
Placing in space	Mise en espace, spatialisation
Play	Pièce (de théâtre)
Playwright	Dramaturge
Poetry	Poésie
Practitioner	Spécialiste, expert
Producer	Théâtre : Metteur en scène Cinéma, TV, Radio : Producteur
Production	Théâtre : Mise en scène Cinéma, TV, Radio : Production
Prompter	Souffleur
Rehearsal	Répétition (théâtrale)
Representation	Théâtre : Interprétation (de rôles) Peinture : représentation
(to) Run	Diriger, gérer un théâtre
Scenario	Théâtre : Scénario
Scene	Scène (en Pratique et en Abstraction) Décor
Scenecraft	Scénographie
Scenery	Décors
Scenographer	Scénographe
Screenplay	Cinéma : Scénario
Screenwriters	Scénariste
Setting up and installation	Aménagement
Silent play	Pièce muette
Sketch	Art : Esquisse, dessin à grand trait Théâtre, TV : Sketch

Abstract

This thesis will investigate some notions related to the artistic domain such as creativity and improvisation. We will explore the theatre direction to get a precise idea about what this field offers. Then we will look at the subject of the virtual reality, trying to detect its possibilities and its boundaries when it deals with the artistic domain. To be more practical, we will present the Software *Visual Assistant* and the requirements of theatre students and theatre directors. After a study of the programming environment and a review of software reuse techniques, the programme designed in this work will also be presented.

Résumé

Ce travail va examiner une série de notions liées au domaine artistique telles que la créativité et l'improvisation. Nous allons mener notre exploration en direction du théâtre afin de se faire une idée précise de ce que ce domaine offre. Ensuite nous allons nous intéresser au sujet de la réalité virtuelle et tenter de détecter ses possibilités et ses frontières lorsqu'elle est appliquée au domaine artistique. Pour être un peu plus pratique, nous allons présenter le Logiciel *Visual Assistant* ainsi que les besoins des étudiants en art théâtral et des metteurs en scène dans le domaine théâtral. Après une étude de l'environnement de programmation et quelques rappels de techniques de réutilisation de logiciels, l'application conçue dans le cadre de ce travail sera présentée.

To the memory of Éric,
the first graduate of the family as an industrial engineer.

— Vincent

Table of Contents

Acknowledgements

Many of my friends have graciously agreed to review specific chapters for content and/or style. In addition, many of other friends have made numerous critical suggestions. Here is an alphabetical list of them : Alain Jossart, Christian Willems, Clémentine, Didier Roland, Elisabeth Gilen, Madalina, Moussa Wahid, Nora Condon, Olfa Lamouchi and Roberto Giglioli.

I should not forget the contribution of Dominique Serron, theatre director, and Pascal Georis, lighting technician. I thank them for their explanations about the theatre field.

I want to express my gratitude towards my supervisors Professor J. Berleur S. J. and Professor C. Beardon, at the University of Art and design of Plymouth, for their understanding and support.

There are also a lot of people that I have not cited above but who helped me in various ways. I thank all of them too.

Conventions used

Italics are used to indicate new terms, quotations and to put the stresses on different components on a field or even on important words in close relationship.

`Courier font` is used for programming codes.

Square brackets [ ] are used to add some missing word in quotations.

(s)he is used for *he* or *she*

He/She is used for *He* or *She*

him/her is used for *him* or *her*

Introduction

This work is composed of two sections. First, we will explore the different notions that can be encountered in any artistic domain such as creativity, improvisation. Then we specialise our analysis and explore the domain of theatre directors to understand the different components relevant to this profession.

Our interest in the theatre director field is legitimated by the fact that the software visual assistant is intended, among others, to be used by them. Our aim here is to understand this domain, so the software can response as much as possible to the needs of the end users.

A chapter is developed about the virtual reality. The objective is to find out the possibilities and the boundaries of this reality when dealing with the artistic domain in general, and particularly with the theatre field.

The second section is more technical i.e. the stress is put on the programming side of the work. Indeed we start by a subsection dealing with Visual Assistant and the needs of the end users of this software and analyse in which way the software responses to the requirements.

Another subsection deals with the reusing of existing knowledge. It is an opportunity and a learning experience for us to have a glance to the existing tools and methods and to place our works among what exists. The work performed is based on a Mac version designed in C language and is intended to produce a PC version of the software Visual Assistant designed in C++ language.

We wend by the architecture of the program designed in this work, while explaining the major differences between C and C++ languages and the reason underlying the choice of C++, and not another language such as Java.

State-of-the-art

Over recent years there has been much interest in the relationship between computers and theatre. In this section we will introduce some developments of the digital technologies in the theatre field: on-line archives of theatrical materials, virtual actors, use within performance, assistance for directors and assistance in theatre education.

On-line archives

Hence archives of play texts, critical essays and review articles are, so far, the richest resource for theatre within Internet. These sites have had time to evolve.

Among the most useful play archives, there is the Shakespeare Archive[1] at the Massachusetts Institute of Technology. Of course, we have a lot of other Shakespearean sites.[2] There is the possibility to access all Shakespeare's plays on-line. In addition, each play is presented as a hypertext so the user is able to search for particular words or phrases and learn of their background. Users can also participate in discussion groups and find out other useful information on Shakespeare and performances of his plays.

Internet is also used to publish the work of less known living playwrights: it has become a medium for them to advertise their work and perhaps get their particular genius discovered by a theatre team or even a theatre manager.[3] Other theatre professionals like designers, actors, scenographers... are increasingly advertising themselves on Internet.[4] More, Internet is a general marketing tool for (local) theatres.[5]

---

[1] http://the-tech.mit.edu/Shakespeare/

[2] http://www.shakespeare.com

http://www.rdg.ac.uk/AcaDepts/In/Globe/home.html

http://www.gh.cs.su.oz.au/~matty/Shakespeare/

[3] The Dramatic Exchange at http://www.dramex.org/

ELAC on-line Plays at http://www.perspicacity.com/elactheatre/index.html

Screenwriters and Playwrights Home Page at http://www.teleport.com/~cdeemer/scrwriter.htm

[4] The British Actors' Register at http://www.internet-ireland.ie/power/actor/actor.htm

Richard Finkelstein's designs at http://www.artsozoo.org/rf/

On-line performances

Though there has been some use of Internet in live performance. For example, *George Coates' performance works*[6] uses real time Internet audience participation in him theatre performance.

Another work is the *Daisy's Amazing Discoveries*[7] (1996), that uses Internet to introduce professional actors, by adapting them to the limitations of the current on-line technologies, to make audiovisually and spatially oriented drama. A slideshow with actors' voices is one of the many media Daisy uses.

David Blair and Tom Meyer (1997) have explored the possibility to tell stories in multi-user three-dimensional environments incorporating synthetic actors. This involves developing an Internet site, which can provide users and readers with text and still and moving images related to a story in the order they request. It can also provide three-dimensional environments in which to explore the narrative and the ability to conduct dialogues with other users/readers.

Talent Source at http://www.talent.com.au/ts/menu.html

[5] Theatre Central at http://www.theatre-central.com/

[6] http://www.georgecoates.org/

[7] http://www.coronet.fi/daisy/

Director's Assistant and Production Assistant

The *Director's Assistant*[8] application provides a software environment in which a director or a playwright can create various dramatic elements through which the dramatic whole of a play to be staged can be analysed and pre-designed. As the elements and their relationships to each other are defined, the software will also be capable of producing a variety of visualisations on the structure of the play under the pre-design process. More elements and their contents are introduced and networked with each other, allowing the user to try different approaches.

In order to bring the conceptual ideas into actual visualisations and into the final realisation, and to facilitate even more real learning situations, the Director's Assistant can be used together with the *Visual Assistant*[9] software, the software of this dissertation, and the *Production Assistant* software. Production Assistant is basically a management system for controlling, booking and organising the required resources of a drama performance, planning production schedules and marketing and overviewing the representation information from different viewpoints. For example, from the viewpoint of a theatre, a theatre director, or even a stage designer.

Scriptwriter's Assistant

The scriptwriting software, called *Scenario*[10], allows the user to develop a simply sketched-out idea into a play, a short or full-length movie script, a television series, advertising scenarios, or even documentary film script. Scenario guides users by the means of three series of questions helping them to come closer and closer to their topic, characters and scenes. It is designed for a classroom or a distance learning situation.

---

[8] More information and beta version of the software can be obtained from http://vconf.hut.fi/hamlet/

[9] It helps in the visualisation of potential set designs by bringing together a variety of newly created or archival material within a simplified stage context.

[10] Developed by SU.MA.FA Productions Ltd, Paris.

A What is (Theatre) Direction[1] ?

In this chapter, we will explain in the first step the notion of *creativity* and mention some comments from many authors and scientists. We will try to present an account for the way of this strange feeling that is the creativity, especially in the field of theatre. Most of this first part is based on the work of Margaret Boden[2]. At once, we will relate a definition given by Magaret Boden to creativity and a second definition derived from the dictionary the Petit Robert about the subject. The aim is to get an idea about the notion of creativity. After that, we will bring into the scope the remarks of some authors who said creativity is an unconscious process, an intuition. Furthermore, we will define Poincaré's four phases of creativity to establish the link between

---

[1] In English, we can use *(theatre) production* or well *(theatre) direction* for the « mise en scène ». The two generate some confusion because they have other meanings in the movies, television and radio fields. (See appendix *French Translator*) We have chosen *(theatre) direction* for this work.

[2] Margaret Boden, *The Creative Mind*, Abacus, London 1992.

the domains of Arts and Science. Is it possible to have the same sense in the two domains? Next, we will define two different meanings of creativity, one historical and the other psychological, to put emphasis on the importance of a discovery.

In the second step, we will define and explain what is *improvisation*. At the same time, we will establish the difference between *improvisation* and *drama*[3].

In a third step, we will come to the subject of this section, which is the *theatre direction*. We will illustrate the work of the *theatre director*[3]. We will try to define the task of the theatre director and how to relate it to the other components like *scene-craft*[3], drama, theatrical aesthetics. Moreover, we will explain the different steps to make a *play*[3] from the *casting*[3] until the *dress rehearsal*[3].

Finally, we will see the basic theatrical aesthetics of the theatre direction. This illustration will help understanding the domain of application of the *Visual Assistant*[4] software. In fact, we prefer to define the theme of the software, because if those notions are not previously defined, it is not easy to get the essence of this work. We think that after that, we have in hand all we need to become imbued with the job of theatre directors.

---

[3] See appendix *French Translator*

[4] That is the developed software in this work. The Visual Assistant software aims to support theatre people, especially the theatre directors, by *sketching* (See appendix *French Translator*) the visual tasks in general.

1) Where do brainwaves[5] come from?

As said before, our starting point will be a trial of definition(s) of creativity. The task is all but easy.

> Margaret Boden defines creativity as the capacity " *to bring into being or form out of nothing.* "[6]

If we interpret this strictly, creativity would come out from nothing, which is impossible in our standpoint.

Colin Beardon provides us with another definition of creativity:

> " *The search for meaning or for the expression of meaning and the meaning therefore cannot pre-exist the practice.* "[7]

If we interpret strictly his definition, creativity would not come from nothing, and it is reduced here to a question of meaning. Furthermore, it is impossible before practising to express the meaning. How does Colin Beardon define *meaning*? Does he consider that everything has a meaning? Unfortunately, these questions will remain unanswered; we did not find any clarification about the notion of *meaning* in any of the author's articles.

---

[5] See appendix *French Translator*

[6] Ob. Cit., Margaret Boden, *The Creative Mind*, Abacus, p. 1

[7] Colin Beardon, *The design of software to support creative practice*, in: *Proceedings of IDATER 1999*, Loughborough University 1999, p. 4

We can also look at the definition of a dictionary[8] to have another point of view:

> " *Power of creation, invention: inventiveness.* "[9]

To understand this sentence, we have to look for the meaning of *creation* and *invention.*

For the first one, i.e. creation, we have some different definitions, depending on the context:

- " *Term of religion: Action to give the existence, to rise from nothing.* "[10]
- " *Action to make/do, to organise a thing that did not exist before.* "[11]
- " *Term of physics: Creation of pairs: materialisation.* "[12]

In the case of the second one, i.e. invention, we can also see some different meanings:

- " *Didactic: Action to find.* "[13]
- " *Action to create or to discover.* "[14]
- " *Imaginary thing, discovered thing.* "[15]
- " *Special word of the Arts language (technical, critic, historic, etc): ability to build in the imaginary.* "[16]

---

[8] Le petit Robert

[9] « Pouvoir de création, d'invention : inventivité », Le Petit Robert.

[10] « Terme de religion : Action de donner l'existence, de tirer du néant », Le Petit Robert.

[11] « Action de faire, d'organiser une chose qui n'existait pas encore », Le Petit Robert.

[12] « Terme de physique : Création de paires: matérialisation », Le Petit Robert.

[13] « Didactique: Action de trouver » , Le Petit Robert.

[14] « Action de créer ou de découvrir (qqch. de nouveau) », Le Petit Robert.

[15] « Chose imaginaire, inventée » , Le Petit Robert.

[16] « Mot spécial au langage des Arts (technique, critique, histoire, etc): Faculté de construire dans l'imaginaire » , Le Petit Robert.

Now, we can analyse what the dictionary tries to tell us. First, we need power, i.e. ability to have creativity and inventiveness. The use of the word *power* may suggest that not everybody can do it and also that, if a person has that capacity, it is neither endless nor permanent.

What is creation? As Margaret Boden, the dictionary gives the same meaning in the two first parts of its definition. But, it adds in the third part the *capacity of materialisation*. In other words, creation is something that could be material or immaterial as well. We notice that the definition given by the dictionary enlarges the scope of creativity to make it encompass other kinds of creativity that were not taken into account by the definition given by Margaret Boden.

What is *invention*? Is it the same as *creation*? *Invention* is the ability to build in the imaginary[17], allowing us to go to the unreal world. We can say that *creation* and *invention* are very similar. Moreover, the definition of the dictionary about the creation talks about an idea of discovery from nowhere by the means of a certain power. But, is it possible that this power comes from nowhere? In the same way, is it possible that discoveries come from nowhere, with no previous acquisitions coming from the past experiences?

By comparing the definitions of Colin Beardon, Margaret Boden and the definition given by the dictionary, we can conclude how difficult is the task that we are undertaking, i.e. defining the creativity and its origins. This task seems to be more difficult than what we can think, no *consensus* is reached about the definition of creativity.

Indeed, let us interview several people about creativity, for example ten people, it is probable to get ten different definitions, and if we will go further and ask these ten persons to judge if some listed ideas are creative or not, we will be surprised of the result and certainly, there will be disagreement among this group whether an idea is creative or not.

---

[17] See *invention*'s definition given from Le Petit Robert.

We cannot deny that creativity exists and that it happens sometimes but the question is how does it occur? Is it a *mystery*? Or is it something that science can explain? To be more precise, the question can be divided into two parts, the first one will be asking how does creativity happen and the second one being what makes it occurring? These two questions guide us to look at two different sides of creativity. The way creativity happens is a puzzle, so it can be explained in scientific terms and as Margaret Boden said:

> *" Scientists take puzzles in their stride. "*[18]

The major issue is to know where creativity comes from. It is a mystery and mysteries are beyond science.

All the definitions listed above do not provide us with a satisfactory answer about the origins of creativity. But, what if we explore the track of intuition as origin of creativity?

a) Unconscious process, intuition

Many researches indicate that creativity is not a matter of imaginary. They simply recall the sudden appearance of the solution to a problem that individuals had been working on with no apparent success.

The suddenness of the solution is not its only strange feature. The answer to the prior question may be of an unexpected kind. Picasso, for example, implied that he did not form any expectations, that he could advance his art without having to look where he was going:

> *" I do not search, I find. "*[19]

---

[18] Ob. Cit, Margaret Boden, *The Creative Mind*, p. 1

[19] « Je ne cherche pas, je trouve », Picasso.

We can also see that genius people are the same as we are. In fact, they have developed this sense of creativity more than normal people have, and this looks like magic for *usual* people.

Einstein notes that Mozart was " *only a guest on this earth* "[20], but declared also: " *Others may reach heaven with their works*[21]. *But Mozart, he comes, he comes from there.* "[22]

From the creation point of view, intuition is an enigma. Sometimes, it is experienced as a sudden flash of insight, with no immediately preceding ideas in consciousness. Hadamard[23] is a case in point:

" *On being very abruptly awakened by an external noise, a solution long searched for appeared to me at once without the slightest instant of reflection on my part.* "[24]

Insights do not come from gods, and they do not come from nowhere, either. Flashes of insight need prior thought processes to explain them. If novelty is grounded in prior ideas, can it really be novelty?

Coleridge[25] regarded the unconscious as being crucial in the creation of *poetry*[26]. His poetic vision of Xanadu[27] comes to him in an opium-induced reverie. In this case, the new ideas were fleeting, and easily lost through distraction.

---

[20] W. Hildesheimer, *Mozart*, London 1983, p. 15

[21] See appendix *French Translator*

[22] Ibid., p. 15

[23] (French: 1865-1963) mathematician who had a great influence on the mathematical French school at the beginning of this century.

[24] Koestler A., *The Act of Creation*, Picador, London 1975, p. 117

b) Poincaré's four phases of creativity

In the same way, Poincaré[28] suggested that creativity requires the hidden combination of unconscious ideas. He distinguished four phases of creativity:

- *Preparation*:

  It involves conscious attempts to solve the problem, by using or explicitly adapting familiar methods. There is often no apparent success. Seemingly, as the experience is unproductive, it has also a frustrating taste.

- *Incubation*:

  The conscious mind is focused elsewhere, on other problems, other projects, or even on a sightseeing trip. The fruitful novelties are initially generated. Poincaré said that below the level of consciousness, ideas are being continually combined with a freedom denied to waking, rational thought. He insisted that this phase includes productive mental work, not simply a refreshing rest.

- *Illumination*:

  After that, comes the flash of insight. Despite its unexpectedness as a conscious experience, Poincaré ascribed a significant mental history:
  " *sudden illumination* [is] *a manifest sign of long, unconscious prior work* "[29].

- *Evaluation*:

  Finally, deliberate problem solving takes over again. As the new conceptual insights are itemised and tested.

---

[25] (English: 1772-1834) Poet, philosopher, dramatist, translator, journalist, preacher, critic, theoretician of the religious, of the culture and the State. He is a great person of the English romantic literature revival.

[26] See appendix *French Translator*

[27] Name of the dream palace that Coleridge evoked in his poem *Kubla Khan*.

[28] (French: 1854-1912) Engineer and PhD in mathematical sciences. He was professor of the university of Caen, and after this, of the University of Paris. At thirty-three years old, he was member of the French Academy.

[29] Henri Poincaré, *The foundations of sciences: science and hypothesis, The Value of Science*, Science and Method, Washington 1982, p. 389

Without ignoring the role of consciousness, Poincaré insisted that

> " *unconscious work is possible, and of a certainty it is only fruitful, if it is on the one hand preceded and on the other hand followed by a period of conscious work.* "[30]

In the domains of mathematical and/or scientific creativity, Poincaré's account is well suited. Particularly, in the case of specific problems that were explicitly identified and explored during the preparation phase and used as a test in the evaluation[31]. But it is not always that way in artistic creativity. The artists may have no clear goal in mind. Moreover, painters, composers and theatre directors spend many hours evaluating their brainwaves. They sometimes imply that no such reflection is involved. As quoted above, Picasso said: « *Je ne cherche pas, je trouve.* » This is not to show that Picasso did not use no evaluation: How did he know that he had found it? When did he find it? It shows only that, in some cases, he judged that the novel structure did not require any modification.

Complete illumination of this kind is comparatively rare. Composers usually add corrections to their manuscript scores; art historians constantly discover the rejected first thoughts of the artist, hidden under the visible layers of paint; theatre directors try some different positions of *characters*[32] on the *stage*[32], change the colours of the costumes in relation with the *scenery*[32] and the *footlights*[32], etc. In summary, the theatre director adapts the stage directions in all the process of rehearsals.

---

[30] Ob. Cit., Margaret Boden, *The Creative Mind*, p. 20

[31] Normally, Poincaré said of *verification* and not of *evaluation* phase. In fact, it's more opportune to speak of *evaluation* in the domain of the art, the term *verification* being more suited for the scientific domains.

[32] See appendix *French Translator*

In short, Poincaré's four phases theory allows us to draw a conclusion about the achievement of innovation in the artistic and scientific domains. Indeed, *arts* and *sciences* achieve their innovations in broadly comparable ways.

What is the unconscious work, and why must it be preceded and followed by consciousness? Poincaré's answer was that preparatory thinking activates potentially relevant ideas in the unconscious, which are unknowingly combined there. A few are insightfully selected because of their aesthetic qualities, and then refined by conscious deliberation.

Obsessive ideas are sometimes so outlandish, and perhaps also disorganised, as to be judged mad. Certainly, the dividing line between *creativity* and *madness* can be unclear. Aristotle said that no great genius has ever been without madness, and Charles Lamb[33] wrote to his friend Coleridge:

> *" Dream not, Coleridge, of having tasted all the grandeur and wildness of Fancy, till you have gone mad. "*[34]

But very often we can tell the difference. For instance, a *" schizophrenic word-salad "* can have the both interpretations: the *creativity* and the *madness.*[35] But it shows no psychological structure comparable to Poincaré's four phases, and it rarely produces ideas which others recognise as creative. At most, it may provide cues triggering someone else's creativity.

---

[33] (English: 1775-1834) He was born in the strange district of the Inner Temple in London. He was poet and essayist.

[34] Livingston Lowes, *The Road to Xanadu: A study in the ways of the Imagination*, Constable, London 1951, p. 498

[35] With full of surprises. Some psychiatrists can find that is creative.

We can also see Koestler[36]'s remark. He said:

> " *The history of human thought is full of triumphant Eureka, but only rarely do we hear of the anti-climaxes[37], the missed opportunities, which leave no trace.* "[38]

Even an exciting idea can turn out to be a dead end. Still others are discarded in error, their creator being blind to their significance. The mistake may be recognised later by the creator, or it may not.

Poincaré attempted to answer this question by appealing to the creator's aesthetic sensibility:

> " *All the combinations* [of ideas] *would be formed in consequence of the automatism of the subliminal self, but only the interesting ones would break into the domain of consciousness. And this is still very mysterious. What is the cause to pass the threshold, while others remain below? Is it a simple chance which confers this privilege? Evidently not... What happens then? Among the great numbers of combinations blindly formed by the subliminal self, almost all are without effect on the aesthetic sensibility. Consciousness will never know them; only certain ones are harmonious, and, consequently, at once useful and beautiful. They will be capable of touching the special sensibility of the geometer* [or other person]. "[39]

---

[36] (Hungarian: 1905-1983) Talented novelist, essayist, famous journalist and dreaded lampoonist. He belongs to the category of writers who have renounced to their mother language for writing in English.

[37] i.e. deceptions.

[38] Ob. Cit., Koestler A., *The Act of Creation*, p. 217

[39] Ibid., pp. 391-392

The citation above describes the experiences of many creative people, including Poincaré himself, and it allows for mistaken insights. But it does not tell us what features make a combination seem *harmonious*, still less *useful*. Nor does it tell us what sorts of combination or transformation are likely to be promising, or how their promise can be intuitively caught.

These questions about the origin and recognition of insightful ideas can be answered only after being able to clarify the concept of creativity, only after distinguishing mere newness from genuine originality. Without a coherent concept of creativity, we cannot distinguish *creative* ideas from *uncreative* ideas. And if we cannot do this, we cannot hope to discover the processes by which creative ideas arise.

People who want to demystify creativity usually say that it involves some new combination of previously existing elements. Haramard, for example, wrote:

> " *It is obvious that invention or discovery, be it in mathematics or anywhere else, takes place by combining elements.* "[40]

In general, combination theories identify creative ideas as those, which involve unusual or surprising combinations. Many psychologists assume that the more unusual ideas are, the more creative they are.

---

[40] Ob. Cit., Koestler A., *The Act of Creation*, p. 120

c) Different senses of creativity

From now, we can also see the definition of different senses of creativity. One sense is psychological: *P-creative*, the other historical: *H-creative*. Both are used to define corresponding senses of *creative* (and creativity) which describe people.

- The *psychological* sense concerns ideas, as well in science, in music, in painting and in theatre, etc that are fundamentally novel with respect to the individual mind which had that idea.
- The *historical* sense applies to ideas that are fundamentally novel with respect to the whole of human history.

Although H-creativity is the more glamorous notion, and is what people usually have in mind when they speak of *real* creativity, P-creativity is the more important for our purpose. In fact, with Visual Assistant, we try to help each one to express his/her creativity, with no regards to the previous existent creative works done by the person using the Visual Assistant or the work done by others.

The familiar adulation of individuals' H-creativity underestimates the extent to which discovery is a social process. It follows from all of this that no purely psychological criterion, indeed no single criterion, could pick out what are, by common consent, the H-creative ideas. But this does not matter: in understanding how creativity is possible, P-creativity is our main concern.

P-creativity is crucial for assessing the creativity of individual human beings, their ability to produce original ideas. Ability is a power that is more or less sustained. In other words, a person's creativity, like his/her intelligence, is a relativity long lasting quality.

d) Computer science creativity

To end this first point, we can say that there exists a set of creativity: *art creativity*, *maths creativity*, etc. But what about computer science creativity? Does it exist? How do computer scientists capture the creativity of someone else inside a program?

There are computer programs and there are computational concepts. If a program fails to match human thoughts, we cannot conclude that the theoretical concepts underlying the program are irrelevant. Computer scientists are as creative as anyone else in other fields, and new computational concepts, new kinds of programs and techniques are continually developed.

A program is what computer scientists call an effective procedure. An effective procedure does not need to be *effective* in the sense of performing the task for which it is used: doing an addition, recognising a harmony, writing a sonnet, placing the characters on the stage, calculating the intensity of lights, doing the *placing in space*[41] to express the distance and/or the closeness, etc. All computer programs are effective procedures, whether they succeed in the task, which we would like them to do, or not.

An effective procedure is a series of information processing steps which, because each step is unambiguously defined, is guaranteed to produce a particular result. Given the appropriate hardware, the program tells the machine what to do and the machine can be relied upon to do it. The computer will do precisely what the program orders it to do.

A program can also be defined as a series of rules, for example. The early artificial intelligence workers, who first defined heuristics as effective procedures, developed the related concept of a search-space. This is the set of states through which a problem solver could conceivably pass in seeking the solution to a problem. In other words, it is the set of conceptual locations that could conceivably be visited. *Conceivably* here means *according to the rules*.

---

[41] Or even *spatialisation*. See appendix *French Translator*

Some of the most important human creations have been new representational systems. They include formal notations, such as Arabic numerals, chemical formulae, or the staves, minims and crotchets used by musicians. Programming languages are a more recent example, offering effective procedures of many different kinds. The computer has its own notation and can represent the different models made by the theatre director. In fact, we can think that we do not have more reduction with the computer notation than the *craft document*[42] written by the theatre director.

A question one can ask here: could creativity be based on rules? Let us see what is happening in music and theatre fields for instance. There are rules in music also which define a metrical search space and no one could say that music is not a creative action. Moreover, in the theatre field, we have a panoply of rules that must be respected by the different stage persons: the theatre director must respect the disposition of the characters, the ray of lights, etc; the *scenographer*[42] must respect the size relationship between objects and actors, etc.[43] As music, theatre direction has a metrical search space and no one can say that the theatre field is not creative. To conclude, we can say that computer science could be as creative as any other field.

On the other side, one can ask whether constraints and creativity could exist together, especially in the context of computer programs. People speak in terms of *rules* and *constraints* in the way of making the software. The critics say that these rules and constraints must be irrelevant to creativity, which is an expression of human freedom. But constraints on thinking are what make it possible. In fact, constraints map from structural possibilities, another one by exploration and modification of the structural possibilities. Whether the added constraints to reach the new possibility are aesthetically pleasing, as opposed to being merely ingeniously productive, is another question.

---

[42] See appendix *French Translator*

[43] See the next section of this chapter: *What is theatre direction?*

If we drop all these constraints to provide new ones, we do not invite creativity but confusion. We do not want to say that the creative mind is constrained to do only one thing. Even someone who accepts all the current constraints without modification would have a choice to do it.

For example, J.S. Bach was constrained, by his own creative decision, to compose a Fugue for the *Forty-Eight* in C minor, so he was constrained to do certain things and not others. But within those musical constraints, he was free to compose an infinitely large range of themes. Similarly, when we speak, we are not limited to say just one thing because of the grammatical rules.

In the same manner, when a theatre director is constrained to make his/her *stage acting*[44], that means that (s)he is constrained by his/her own creative decision to do a thing and not another. But, within the constraints of his/her conception of good theatre direction, (s)he can compose a lot of different manners to *stage-manage*[44] the play, to interpret the drama work, etc.

Margaret Boden invites us to explore another side of computer science. Programs could help explaining creativity, human thought processes, and the mental spaces they inhabit, are largely hidden from the thinkers themselves. The sort of thinking that involves well-structured constraints can be better understood by comparing it with problem solving programs, whose conceptual spaces can be precisely mapped. Imprecise thinking, such as poetic imagery, can be understood computationally too.

The point of interest of Margaret Boden is important, sure, but is not the issue here. Indeed, we are interested to another aspect, that consists in what sense programs can modelise the creativity of someone and in which sense they can help to catch the imperfect sides of the product and help to enhance its quality.

---

[44] See appendix *French Translator*

Sure, by modelising, the theatre director could save money and time. But what we want is a little bit more than that. It could be how the programs could help him/her to express and visualise what (s)he intends to perform.

By the means of the computer, we want to provide the theatre director with a tool permitting him/her to put out the maximum of problems when (s)he realises a new play. This with the aim to refine the raw ideas, to catch the imperfections and to enhance the quality of what (s)he intends to produce, with a maximum of computer assistance. This assistance is not intended to provoke a degradation of his/her creativity.

2) What about improvisation?

Frost says that " *improvisation is fundamental to all drama* "[45].

Drama in its common usage suggests the hegemony of the prescribed script, i.e. the written and the spoken word, but it is now universally accepted that theatre is a matrix of many things, which are called languages in the field of theatre: colour, sound, movement, *staging*[46], etc.

Robert Frost[47] and A. Yarrow define *improvisation* in theatre as "... *the skill of using bodies, space, all human resources, to generate a coherent physical expression of an idea, a situation, a character (even perhaps a text); to do this spontaneously, in response to the immediate stimuli of one's environment, and to do it à l'improviste: as through taken by surprise, without preconceptions.*

Where improvisation is most effective, most spontaneous, least 'blocked' by taboo, habit or skyness, it comes close to a condition of integration with the environment or context. And consequently (simultaneously) expresses that context in the most appropriate shape, making it recognisable to others, 'realising' it as act. In that sense improvisation may come close to pure creativity... "[48]

---

[45] Forst A. and Yarrow R., *Improvisation in Drama*, Macmillan, London 1990, in the opening pages.

[46] See appendix *French Translator*

[47] (American: 1874-1963) He was poet and writer. His work has a important place in the American poesy of this century.

[48] Ob. Cit., Frost R. and Yarrow A., *Improvisation in Drama*.

Improvisation is probably one of the most important creative features that have been (re)discovered this century. In the history of theatre prior to our modern area, improvisation was usually relegated to the techniques of low comedy forms.

It is partly through the development of the avant-garde and jazz in the twentieth century that it has become a prominent feature of all live arts. Eminent modern theatre *practitioners*[49] such as Constantin Stanislavski[50] and Vsevolod Meyerhold[51] used improvisatory techniques in their rehearsal processes. Antonin Artaud[52] saw it as central in the making of theatre:

> *" The ideal of a play built right on the stage, encountering production*[49] *and performance*[49] *obstacles, demands the discovery of active language, both active and anarchic, where the usual limit of feelings and words are transcended. "*[53]

It was not, however, until the work of Jacques Copeau[54], that improvisation came to be used as a central creative element in the training of actors. Improvisation is now well established as the starting point of all western theatre training, in every kind of educational institution from the conservatoire to the university.

---

[49] See appendix *French Translator*

[50] (Russian: 1863-1938) He was actor, an important theatre director and theatre manager. Moreover, he was a reforming theoretician of drama and lyric arts, and his work has influenced the theatre field.

[51] (Russian: 1874-1940) He was the champion of anti-realistic theatricality and a more inventive theatre director of the twentieth century. His work has extended the perspective of theatre direction to the infinite.

[52] (French: 1896-1948) He was in chronological order of his life: poet, actor, scenographer and theatre director. His works have influenced the second half of the twentieth century.

[53] Artaud Antonin, *The theatre and Its Double*, Caulder and Boyers, London 1970.

[54] (French: 1879-1949) He was linked to the literature of his period, the first general manager of *la Nouvelle Revue Française, drama critic* (See appendix *French Translator*) of *l'Ermitage* and *la Grande Revue*. As he knew the loss of credit of theatre at his time, he lived with the wish to squeeze together *theatre* and *poetry*. In fact, there were a lot of confusions between the two of them in his century. His effort was to refine the techniques and the customs on the stage.

Improvisation has become the ground or basis of the creative process of training *performers*[55]. Every performer needs to understand and to gain the skill to improvise, be it freely or within a rigidly prescribed text. Improvisation is the vehicle through which the student actor develops his creative technique, it is a rehearsal tool, it is the means of making new texts and, finally, it is the factor, which signifies all *live theatre*[55].

We have discovered another dimension in the artistic field: *improvisation*. Our question is: can a program encompass this dimension? It is not easy to answer such a question, especially if we think of improvisation as the act of the actors and not of the theatre director to whom the program is designed. But if we think of improvisation as something that the theatre director can do, our answer to the question above will be *positive*.

[55] See appendix *French Translator*

3) What is theatre direction?

a) Scenecraft, Drama, Theatrical Aesthetics and Theatre Direction

The subject seems simple and very restricted. But, if we think about it more, we can see that, in reality, it is very complex and with an infinite area. For most people, it is summarised by a material question. The theatre direction is reduced to the good or bad performance of a drama work[56], to the exuberance of the dresses, and to the number of *walk-on actors*[57]. We can consider that *theatre direction* and *scenecraft* are in close relationship, but are very different:

- *" Scenecraft is only one part of the direction. It involves the setting up and installation*[57] *of theatre materials and the representation*[57] *in perspective;*
- *The role of the theatre director is to take charge of the* placing in space *and, also to transmit the essence of the work*[57]*, to have harmony between the actors, objects, music, lighting, colours, costumes and the original text. "*[58]

---

[56] In Art, the term *dramatic* has not the same meaning. It is the set of the theatrical activities, generally seen at the professional level.

[57] See appendix *French Translator*

[58] « *La* scénographie *n'est qu'une partie de la mise en scène. Elle s'occupe des aménagements matériels du théâtre et de la représentation en perspective. Tandis que le rôle du metteur en scène est de s'occuper de la* mise en espace *(spatialisation); mais aussi de diriger l'*intention de l'œuvre, *avoir une harmonie entre les acteurs, les objets, la musique, le son, l'éclairage, les couleurs, les habits, et le texte original.* » A conversation we had with Dominique Serron, theatre director.

But, out of these considerations, we can see that the theatre direction merges into the drama work. We can consider that we have two different notions:

- On one side, the *drama*, i.e. the own work of the poet, the dramatist.
- On the other side, the *direction*, i.e. the common work of all those who converge to the performance, with regards to any degree.

These two different fields of Art can be in close relation. When a *playwright*[59] thinks about the theatrical dispositions that they do not come necessarily from the personality and the passion, (s)he makes *theatrical art*[59]. When a comedian throws some emotions into relief, to which the work's author has not given a sufficient importance, (s)he does *drama*.

Although the limit between the two arts is not real, we do not examine drama, but theatrical art. It is just for making the subject easier.

We try to explain the general aspect that the performance must have and the determination of the special effects of the acts, in which the play is decomposed.

We want to discuss the development of the theatre direction of a play and its theatrical aesthetics, but not about the theatrical architecture, or the decorative painting, the difficult perspective science, the programming of the set of light with the computer, the placement of the scenery, etc.

The director must transpose the original work, and his/her role is to direct the *audience*[59]'s attention towards the world of the author and on the emotions that (s)he wants to communicate. We will just be interested in this most important aim of the direction, which is to enable a good presentation to produce a maximum of feelings for the audience, which would be amazed by the beauty of the work. We have therefore the director's interpretation of the work. Of course, the director knows the general emotions of a particular author, but (s)he can never retransmit the same emotions or ideas of another person entirely.

---

[59] See *French Translator*

" *Theatrical aesthetics is the study of principles and the general or particular rules, which govern the performance of drama work and resort to the production of emotions and beauty.* "[60]

Finally, we must not forget the most important work of the director: *human relations*. Good direction depends on the goodwill of everybody. Each person has its merits, which must be valued and developed as much as possible. The director must direct everyone under his/her responsibility as (s)he wants.

" *The artistic part is fundamental, but there are many other things that must be looked at such as the* organisation *of the play and especially* communication *so that the team and the actors achieve what is expected from them.* "[61]

b) How does the foundation of a play work?[62]

The role of the director is to adjust at the best the different steps and components of a play. We can compare the director to a *conductor*[63]: both must take into consideration the aspects as the space stage, the needed material, the labour, the light, the scripts and the period. In fact, a play is a real co-ordination work. The director is not only the person who manages the place of the objects and the actors, but also the co-ordinator of what is the play. And we can say that the play is as a *meticulous Swiss watch*: things are left to chance as less as possible.

---

[60] Becq de Fouquières Louis, *L'Art de la mise en scène : essai d'esthétique théâtrale*, Entrevues, Marseille 1998, reediting of 1884, p. 10

[61] « *La partie artistique est fondamentale, mais il y a beaucoup d'autres choses qu'il faut regarder comme l'organisation de la pièce, mais surtout la communication : faire ressentir à l'équipe, aux acteurs ce que l'on attend d'eux.* ». A conversation we had with Dominique Serron.

[62] Based on conversations with Dominique Serron and Pascal Georis, *lighting technician* (See *French Translator*).

[63] See appendix *French Translator*

Firstly, the theatre director and the *theatre manager*[64] have to reach an agreement in terms of the choice of the text to *run*[64]. Then, the theatre director composes a team: *stage manager*[64], *scenographer*[64], *costumers*[64], *lighting technicians*[64], actors, etc. Once composed, the team will have meetings to co-operate for preparing the drama work.[65]

After that, the director can put on the play by distributing all the roles to the actors[66]. This allocation of the roles is the most important part of the play because the final success depends on it. Each actor studies his/her role and, after a certain time, we cannot change the role between the actors.

While actors are working, the director sends the *specifications*[64] to the scenographer. He/She is a specialist of the study of the theatrical material arrangements. He/She makes the decoration of the stage: the plan of the stage, all the needed objects of the play and if necessary the missing objects. Some of them make first a *model*[64] of the stage, that is a reduction of the real stage, to have an idea about the decoration. This way, it is easier to speak to the director and to meet his/her requirements. We have a dialog between the director and the scenographer, and the material must meet the script as much as possible. Moreover, all objects must correspond to the actors' characteristics.

---

[64] See appendix *French Translator*

[65] All the theatre directors do not always do this co-operation. Many theatre directors prefer *directing* (See appendix *French Translator*). According to Dominique Serron, this co-operation is necessary because the final result depends of the willingness and the participation of every one.

[66] In this work we do not take into account the difference between *comedian* and *actor*. But we can see that they may be different, as Louis Jouvet says: "*The actor can play just some roles because he deforms other roles by his personality, and that the comedian can play all roles. The actor lives in the character, but the comedian is lived by the character*". « *L'acteur ne peut jouer tous les rôles ; il déforme les autres selon sa personnalité. Le comédien, lui, peut jouer tous les rôles. L'acteur habite un personnage, le comédien est habité par lui.* », André Juvet, *Réflexions de comédien*, Sablon, Bruxelles – Paris 1944, p. 137

The size of the stage is very important, because the movements of the actors depend on it. The director must arrange the recited script in relation to the length of the movements on the stage.[67] In fact, if we change the size of the stage, we change also the length of the script as said by the actors. In summary, the script must respect all the space, the objects, and the decoration of the play.

When all comedians have studied their roles, then comes the composition called *rehearsal*[68]. This is a long and meticulous work. The director must adapt the actor's personality with regards to the person's character of the story of the play: such role must be dull and such other role must be accentuated. This job needs a lot of time to reach the perfect harmony between all the actors.

At the same time, the director studies the scenic movements: (s)he determines the different successive places that the characters must fill in relation to each other or to the decoration. Of course, we do not need all the objects of the stage in this kind of rehearsal because the actors are professional, and they have a real sense of abstraction. The direction regulates the entrances and the exits, which can require an arrangement of the script. During the rehearsals, the costumes are made.

---

[67] For Dominique Serron, drama works cannot be modified. We must integrally respect the original message delivered by the drama works. So they cannot loose their sense or even contradict the author's thoughts and brainwaves. Though, when a work's adaptation, in the context of a translation, is concerned, changing paragraphs and words do turn out to be imperative to respect the author's original world.

[68] See *French Translator*

There are two different types of stage given by the theatre director. They not only involve the actors but also everybody else related to the play. More, the latter must take the stage directions given to the actors into account.

- *Essences of work*: the theatre director indicates how a particular character in the play would behave and the actor must then reproduce the features of that character to correspond as much as possible to the director's description.
- *Placing in space*: the behaviour of actors with (theatre) objects in space. We can also see other things in the placing in space, such as the *relation of proximity*. Depending on whether one chooses to place the character far from the audience or close to the audience, one obtains different relations and emotions. For example, in Shakespeare, there is both proximity and distance, and so Shakespearean spirit is both removed from, and in the public, at the same time. A deep stage is therefore necessary which will allow the audience to be both distant from and, close to the theatrical action (and thus the actors): one plays on the depth of the *visual field*[69]. One enters here into a science, that of *drama psychology*.

We are within a drama framework, in which the theatre director can choose how to represent the work, that is, (s)he is free, for example, to choose the objects, the way in which to place the actors, even to choose not to include objects although they appear in the text of the play. If we say: "I will open the window", one could very well react as if the window was there, when in fact it is not.

In this way, we can create reality, in our three-dimensional space. We can create reality as we can create the imaginary, by changing the objects or by not including them even though they are specified in the original text.

---

[69] *Close-up* in cinematic terms.

Also, the director speaks with the lighting technician(s). He must make a special atmosphere with regards to the meaning of the play by the help of colours, special effects of light. Moreover, the lighting technician must act as a guide to the audience in the different successions of the parts of the play. For example, the lighting technician focuses the actor who speaks[70].

Today, to reduce the cost, we have usually one piece of scenery for the whole duration of the play. But, some plays of previous centuries needed a lot of different pieces of scenery. That is the work of the lighting technician to create the illusion that we are in a different area by the help of different sets of light or special effects.

If we have music in the play, we need choirs or/and music players. With music, we can accentuate the feeling of the play, we can simulate an area, etc.

After the partial individual rehearsals, the combined rehearsals come, where all the accessories run the role that they must have, and at last the dress rehearsal is run with the costumes.

Finally comes the day of the first performance, where all is formalised by the theatre director's interpretation of the play. All missing points in the performance, outgoing the traced line in the rehearsal can floor the characters.

c) General theatrical aesthetics of theatre direction

" The intrinsic value of drama work does not always have the measure of the value attributed by the contemporaries "[71]

This sentence cannot have a lot of contradictors. In fact, we can see the unjustified infatuation for such poet or such drama work at a specific period. But now, we

---

[70] Except if it is a desired effect of the play.

[71] Ob. Cit., Becq de Fouquières Louis, *L'Art de la mise en scène : essai d'esthétique théâtrale*, p. 15

know just these authors by the title of their works. The Art works depend on momentary caprices of mode and feeling. But, some of them are eternal.

We can also quote another sentence of the same author:

" The intrinsic value of drama work does not depend on acts representative effect "[72]

It is not by increasing the representative effect of drama work, by the help of theatre direction, that we can increase the intrinsic value.

On the other side, the representative effect is not created by just the theatre direction. All drama works own by themselves a *poetic* value and a *representative* value. Only, in the poet's imaginary, they are often in the reverse relation that we can see in the theatre field, where the theatre direction modifies and sometimes sets down the proportion. We can say that the theatre direction is the blossoming, and delivered visible, of an ideal germ.

Let us mention the opinion of Louis Becq de Fouquières about the ideal theatre direction; illustrating the subjective representation of the drama work:

" The theatre direction is always discreet, it seems, it becomes obliterated, disappears and reappears without asserting itself to our mind or without distracting it; the mobile scenery where everything stays on its place, but where everything moves closer to us or goes away in relation to our imaginary "[73]

But this Fouquières's vision is not possible: the real performance is very different of the ideal performance. When the *stage designer*[74] or the theatre director immobilises all things, which requires precise mobility and fleeting, the real performance deceives in relation to what we had imagined. All the imperfections is accumulated: the

---

[72] Ob. Cit., Becq de Fouquières Louis, *L'Art de la mise en scène : essai d'esthétique théâtrale*, p. 18

[73] Ibid., pp. 23-24

[74] See appendix *French Translator*

imperfect scenery, the imperfect movements, the imperfect diction, the imperfect costumes, etc are an insult to the poetic beauty of the drama work.

However it may be, the ideal performance will always be a model of what the theatre director must have in mind, regarding the intellectual and moral necessities of the audience who (s)he applies to.

Of course, the theatre direction can correct the weakness of a drama work, and by its excess can derive the original attention of the intrinsic value of a drama work. In fact, an abuse of the theatre direction could disturb the audience's judgement of the object, which would be the principle occupation.

When we have some preoccupation on our mind, we can also listen to music, see the objects of our environment. The *intellectual*, *visual* and *hearing* impressions are felt at the same time. The theatre director must play with these three notions, because each of them interacts with the other. He must respect the proper intensity of each sense, that is permitted for each one at a specified time.

We see that the principal aim of arts is to allow *intellectual*, *visual* and *hearing* impressions: the poetry the *intellectual* pleasure, the painting the *visual* pleasure and the music the *hearing* pleasure. But for all three, it is impossible to have a good pleasure, if they infringe on the two other impressions. In the case of the performance of a drama work, the theatre director must respect the intensity of the different impressions. He/She cannot put the intensity to another sense than the sense expected at a certain time. In other words, (s)he cannot disturb the attention of the principal object. As said, the excess of theatre direction can lead the drama into decadence.

But, in another way, we can say that the drama work can require a better theatre direction. Without the help of the scenery, the costumes, the *walk-on part*[75], the number of the different rooms, which can divert our mind, the play cannot face our integral judgement.

We are solicited by the three different senses at the same time. But we fall into the sense that requires the most important place, by doing the loss of the two others. Thus, there is a balance to find in the theoretical art. The art of the theatre director is

---

[75] See appendix *French Translator*

to make the best choice of the manipulation of our three senses in order to remind us the spirit of the drama work.

To respect what we said, we must look for the contradictions between the co-ordinations. We have a lot of contradictions, which come from the imperfect diction of the actors, or well from the poet himself. But, the theatre direction must not add some new contradictions such as the details in the scenery, the discordance between the walk-on actors, or even a costume that does not suit the spectacle.

The contrasts that result neither from the action nor from the adventure of the drama work are the more powerful causes that disturb inevitably the attention of the audience. The theatre direction should never contradict the poetic work, the idea that we can have about the place of the play, the period of the action, the dresses, the habit and the language of the characters.

The theatre direction must respect the drama truth in its integrity. It does not clumsily destroy the relations that the audience create with the characters. The artistic theatre director requires more precaution than audaciousness. He/She must direct the necessary visual impressions to the audience's eyes, and especially put out or attenuate rays too sparkling. By this harmony, (s)he maintains the aims of the theatre direction, and (s)he obtains that by a lot of necessary sacrifices.

The theatre direction is regulated by the rules of human spirit, and we cannot limit it to the specific rules of an academy. But, we do not want to say that drama is a result of coincidence. In fact, luck and Art are mutually exclusive: the first implies a fortuitous conjunction, and the second a preliminary arrangement. And, we can say that:

" *The theatre direction is an Art because it is a fiction* "[76]

[76] Ob. Cit., Becq de Fouquières Louis, *L'Art de la mise en scène : essai d'esthétique théâtrale*, p. 102

d) The theatrical aesthetic around the theatre direction

The stage design[77]

We can see the scenery as an important area of the stage. We can compare the stage design to the painting, staying at a general point of view and without looking at the mathematical aspect.

The scenery must respect the real perspective of the play. When the actors move in the depth, they destroy the illusion of the stage design. But, this consideration of the first order is not important in theatre. The audience accepts that, and when an actor keeps its attention, it does not see the mathematical incoherence between the characters and the stage design. Of course, we must, as far as possible, attenuate the disproportion between the dramatis persons and the stage design.

It becomes evident that the stage designer must avoid the representation of too big objects in the *back stage*[77]. In fact, when an actor arrives at the level of this last plan, we see the disproportion between the characters and the objects of the back stage. Evidently, as we are limited in the depth of the theatre stage, the plan of the back stage represents in reality a plan that is further.

The scientific conditions of the theatre direction and the stage design do not admit all the real possibilities. As any art, the theatrical art has its limits. That is why, the theatre often coasts along the real and the imaginary.

But, the comparison between the painting and the stage design is not perfect: painting represents a moment of an action, but the stage design must be adaptable to the succession of different moments. For the same moment, the painter and the stage designer cannot associate the nature and the human acts in the same manner. The painter can catch the nature with an unfinished movement, but the stage designer will be constrained to finish the movement. The stage designer must reproduce a general impression, just like the theatre director.

---

[77] See appendix *French Translator*

The dresses

The costume is a very important part in the visible sense: it produces our external aspect. We do not recollect the nudity of the persons that we meet.

We can think a costume without the actor or actress. In fact, each time the role will change between the actors, the dress will need a modification. The first rule of the theatre dress is to be in accordance with the age, the stature and the feeling of the actor who will wear the costume. Moreover, the costume must be changed according to the appearance of the actor. The effect to produce by the help of the dresses is immutable, but the means to produce it is alterable.

On the stage, the comedians can modify the costume in function of different acts of the life: the characters have to be natural and true.

The theatre hall[78]

It is impossible that the theatre direction does not take care of the disposition of our theatres and about the defective optical conditions in which the audience is placed. We can see the design of a theatre by this top view: (See Figure 1)

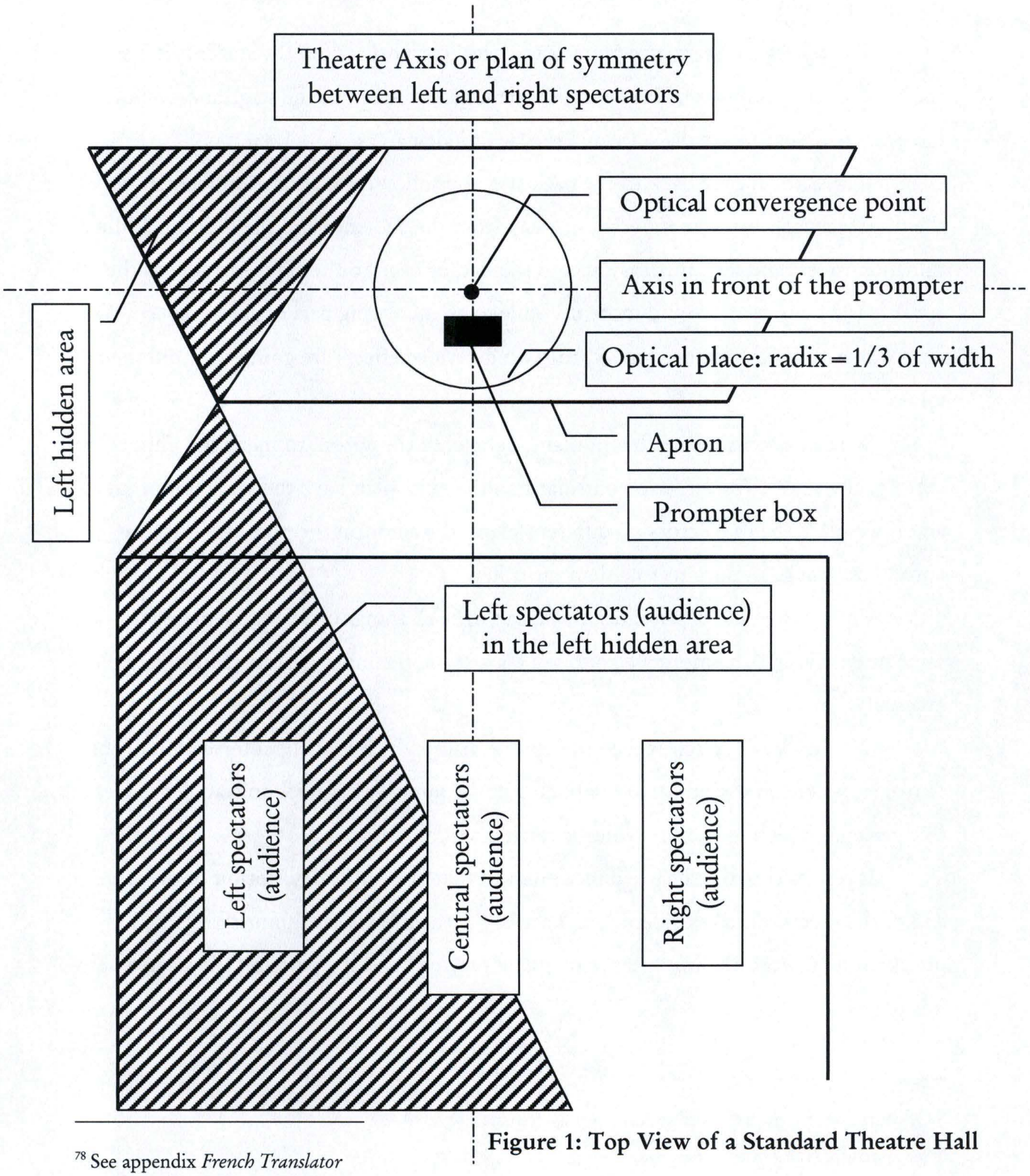


Figure 1: Top View of a Standard Theatre Hall

[78] See appendix *French Translator*

The stage is like a trapezium[79], invariable in the width, but it can vary in height and depth. On the left and on the right, we have the two areas that are more or less invisible: the one on the left to a certain part of the audience on the left, and the other on the right to certain people on the right. The obliqueness of the scenery can change the invisible wide.

The optical convergence point is the excellent position on the stage. It is just in the front of the *prompter*[80]. The central audience are in front of the stage and do not have any problem to see the stage and the two hidden areas.

If an actor moves back to the back stage by following a parallel movement to the theatre axis, every step moves him away from the audience and the footlights illuminate him less and less. If (s)he walks to the left, or even to the right, parallel to the *apron*[80], (s)he comes more and more invisible to an increasing part of the audience. But, if he walks obliquely to the scenery, the two described effects are going to annul themselves.

We can also use the different plans, parallel to the apron, to mark the importance of the actors. In fact, two comedians on the same plan have the same importance, and if we place the two actors on different plans, the actor on the plan closer to the apron has a more important role than the other.

To improve the depth illusion, we can light all things on the same plan with the same intensity, but the more we go to the back stage, the more we can reduce the light gradually.

We can draw a circle, called the optical place, which is at the intersection of the two axes, where all the points are lightened in the same way. The whole audience sees this place, in which the actors' voice is carried without efforts.

It results that the theatre direction must take this short description of the stage and of the theatre into consideration. That is, the theatre direction must keep a good proportion between the importance of a *stage playing*[80] and the place on the stage where it is played.

---

[79] Of course, we have a lot of others kind of stage (circular, squared, etc).

[80] See appendix *French Translator*

In summary, there is always an aesthetic reason in the dénouement that closes together or spreads the characters of the optical place. We can see that the theatrical rhythm follows in its movement the aesthetic rhythm, and the position of the actors is not arbitrary. Of course, the theatre direction must take the position of the fixed objects such as a table, the scenery, etc into account.

The theatre direction can be considered not only as an Art, but also as a science, which is founded on mathematics.

The walk-on part

In a general view, the walk-on part is subject to the rules that regulate the hierarchical disposition of the characters. We must sacrifice the details to the general view. Of course, the theatre director must look at the relative place of each *drama person*[81], but (s)he also must be interested in grouping the walk-on actors and in dividing the walk-on part into harmonious parts, in such a way to have a general effect, and without focusing on an individuality.

When a choir is used to play a passive role, it takes the place of a dramatis person. Moreover, we can say that it is the principal dramatis person[82]. It is important for the theatre director to dominate it in order to improve the impression given by the hearing sense. In fact, the choir can be used to put the spectators into anxiety, or also into the feeling of a special place like, for example a church.

---

[81] See appendix *French Translator*

[82] Ob. Cit., Becq de Fouquières Louis, *L'Art de la mise en scène : essai d'esthétique théâtrale*, p. 176

4) Conclusion

Creativity is a relative object, recovering a wide range of domains. That is the reason why we do not get to a monolithic definition.

The theoreticians aside the origin of brainwaves to various factors: coming out of nothing (Margaret Boden), a matter of creation or invention (Le Petit Robert), due to a certain skill in certain circumstances, based on imaginary, previous experience or not, resulting from intuition, manifesting itself unexpectedly, etc a consensus is far from being reached.

Poincaré's postulates that creativity requires a merely gradual process that consists of a mutual activation of conscious and unconscious work, leading to creative combinations and/or transformations. Innovations though, can only be called creative or uncreative according to a coherent concept of creativity.

What is more creativity has different senses according to whether it is regarding the novel ideas of just an individual (P-creativity) or the whole of human history (H-creativity).

In the field of computer science, creativity is possible. Even if a program is an affective procedure, creativity can be expressed within the rules (of a representational system e.g.), in any domain. Visual Assistant intends to support the individual skills of a theatre director, who can freely modify stage-managing and interpretation while observing the rules constrained by lighting, size-relationship, etc.

Margaret Boden shows us that computer programs can handle well structured as well as imprecise constraints by modelisation. Visual Assistant offers such a modelisation to a theatre director so (s)he can have a general view in advance, which enables him/her to exploit his/her creativity in a more efficient way.

Improvisation can resort to creativity. It is developed itself from a low comedy form to a prominent technique (in all live arts) and practised in every kind of educational institution. Seen as a part of the theatre director's work, improvisation can be programmed. Theatre direction cannot be reduced to just the exterior result of the performance.

Theatrical art covers a more general and essential aspect of the performance and the development of direction, multiple and not leaving anything to chance. A director must synthesise all components (effects, aesthetics, decoration, placement, lighting, etc) into an artistic performance, transmitting as well as possible the original message of the dramatist (according to his/her own interpretation), trying to reach an optimal communication. A theatre director must be extremely versatile.

After the preparatory measures (agreement with the manager, forming of the team of actors, scenographer, lighting technicians, etc), (s)he allocates the roles. Later on, (s)he can specify the taken measures in interaction with the current rehearsals.

A theatre director must take the essences of a work into account, the placing in the space (which makes us enter in the field of drama psychology) within his/her artistic freedom.

His/Her task is to build up a relationship between reality and imaginary. The lighting, scenery and music have their part in this process. Every single thing is to be assembled harmoniously by the dress rehearsal and the light performance.

The theatre director has also an aesthetic role(s). He has to make a work blossom in a way the poetic and intrinsic value speak, in spite of the imperfection of all components. Therefore, (s)he must make best use of the intellectual, visual and hearing effects in the spirit of the drama work.

In brief, co-ordination in theatre is a key to success. Stage design (painting, fixed objects, etc) should do not be too disproportionate to the dramatis persons. The optical condition and construction of the theatre hall have to coincidence with the lighting, the actor's movements and placements, etc.

Art and *Science* converge in the person of the theatre director.

B What is the relationship between real reality and virtual reality ?

This chapter will try to find out the differences between the real world and the virtual world, the limits in representing objects coming from our everyday lives, in some new dimensions that are not so well known to us, but that our imaginary had already dreamed about. These new dimensions are included in the virtual reality.

The major part of this chapter is based on a book[1] dealing with new technologies in the cinematographic world. Seeing that the two worlds[2] are similar in some way, we have adapted the flashing ideas from the already mentioned book, while being conscious and taking into account the differences of the two domains.

---

[1] Frank Beau, Philippe Dubois, Gérard Leblanc, *Cinéma et dernières technologies*, De Boek, Bruxelles 1998.

[2] Cinematographic and theatrical, the last being the subject of this writing.

1) Machines that produce images

We will try in this section to answer some questions that artists themselves are thinking about. We will have three sub-sections:

1. The first one will tackle the question dealing with the relationship between machinism and humanism in the artistic domain.
2. The second will be about the degree of similarity, analogy to the original object included in the images produced by machines.
3. The last one will deal with the question of materiality and immateriality.

Let us try to tackle first in a general way the question of what may be called *the machines of images or pictures*, but with a stress being put on computers. In other words, our interest will be especially on the new technologies.

As far as images are concerned, the term new technologies is used to designate the technical tools related to computer science, which enable someone to produce visual objects.

Everyday we see some bleeding of *luggage* words, that form a chronological list: " *firstly electronic, then synthetic, digital, virtual, cyber and artificial again.* "[3] This historical classification shows the evolution of the terminology of this domain, and shows that we always turn on the two notions that are *immateriality* and *technology*, which are unified to make *computer graphics*[4]. The latter is the combination of computers and graphics, which can produce an immaterial world.

Let us note that computers are not the first machines, in a historical perspective, that are used to produce images. Indeed, we can say that any image requires a technology of production. From its origins, technology is simply an expertise or a know-how. In fact, wood engraving, the Egyptian paintings and classical painting could be considered as *machines of images* that suppose the existence of a device instituting a technical

---

[3] Frank Béau, « Désordre Numérique », *Cinéma et dernières technologie*, De Boek, Bruxelles 1998, p. 47

[4] Computer-aided pictures creation process. See appendix *French Translator*.

sphere to constitute images: it is a know-how that needs at the same time tools (these tools could vary from rules, proceedings to materials) and a functioning (that could be a process, an action, etc).

The use of computers to produce images is not as novel as modern discourse is trying to convince us. Our aim is not to discuss or list the different technologies of images, but we wanted to point out that the problems related to the virtual world or the world produced by means of representations of objects of the real world using computers are not specific to this new technology.

We will try here below, among other to explore the relation between the technique and the aesthetics of the representation.

a) The real, actors and machines

We will start this sub-section by a question:

How can the artistic domain, which is full of humanism, make use of machines?

The use of computers to produce images does not imply that the image is produced by the computer, i.e. that the image produced will remain an image done by a human being and is the result of his/her imagination. It is very important to note that, because it is in this breach that all the humanistic part of art will be lodged: the characteristics of the artist's personality, the character's personality, the sketcher style, the painter's touch, the genius of his/her art, the theatre director's interpretation of the drama work, etc.

So, the machines as tools come between man and the world. Within the system of symbolic constructs, which is the main part of representation, these machines are the intermediaries that use the principle of representation. They will enable someone to represent the objects of the real world as *images*.

At this point of this talk, we can claim that if computers are used as tools, they cannot prevent the artist, the theatre director, etc in any way from his/her genius, nor will they remove the characteristics of the theatre director's personality. We can even say that computers are similar to any other tool i.e. if they are very well used and used for their appropriate functions. They will provide the user with the expected result.

The increase of the complexity not only of the machines but also the tools of representation are, in our sense, due to the fact that we want the virtual reality to be closer to the real reality, even more, we want the two worlds to behave in the same way and more precisely the virtual world is asked to encompass the real world. With a software, in this work the Visual Assistant, we design the system with many symbols of construction that represent an approximation of the reality.

Is the technical progress coming with the machines able to improve the artistic domain? Moreover is the use of machines in the artistic domain a negation of art? To answer this question, we can, on the one hand, state Baudelaire's opinion:

> *" I'm convinced that the badly applied progress of photography has contributed, as every purely material progress, to the degeneration of French artistic genius, which is already rare... "*[5]

Does the machine *coagulate* the ideas of creativity by depriving it of all artistic madness? The answer is very difficult and depends upon the degree of purism of everyone.

---

[5] « *Je suis convaincu que les progrès mal appliqués de la photographie ont beaucoup contribué, comme d'ailleurs tous les progrès purement matériels, à l'appauvrissement du génie artistique français, déjà si rare ...* », Baudelaire, « Le public moderne et la photographie », in: *Salon*, 1859.

On the other hand, we can also consider Grary's opinion:

> *" There is a theoretical cut more decisive than the illusion of continuity what could do to think the technical existence of the camera obscura[6] that can see the connection between pictorial likeness and photographic figuration. That amounts to demonstrate, on the one hand, that the evolution of the mechanisation (i.e. the story of the technologies) and, on the other hand, the question of the humanism and the artisticity (i.e. the esthetical question), are two things very different. The growth of one is not necessarily correlated on the progression or the regression of the other one. "*[7]

According to Grary, the technologies and Art have no influence on each other, but correlation between them could exist. Thus, for his part, Grary does not lock the new technologies as Baudelaire did it.

If we have a look at the history of the machines of images, we can note that there are different kinds that we will enumerate here below. Our aim is not to relate the panoply of machines of images, but the objective is to try to understand the specificity of computers as machines of images among a range of others. (See Table 1 on next page)

The first phase will be called *catching*. An example of this machines is the camera obscura, which is a machine of *pre-configuration*, i.e. it organises the gaze, reproduces, mimes, controls the visual perception of the human eye.

---

[6] L'invention de la camera osbcura (chambre obscure dite aussi chambre noire) comme moyen de reproduction d'une image et son exploitation perspective remonte au temps d'Aristote. En 1611, la chambre obscure qui était une salle est devenue un appareil portatif réalisé, semble-t-il, par un moine allemand Johan Zahn. Niepce en 1826 reprendra l'usage de la camera obscura avec une plaque sensible pour fixer l'image reçue, comme l'avait déjà fait l'alchimiste arabe Getel qui, dès le XVIe siècle, utilisait du nitrate d'argent. L'appareil photo était né.

[7] J. Crary, *Technique of the Observer. On Vision and Modernity in the Nineteenth Century*, Cambridge, Mass., MIT Press, October Book, 1990.

The second phase, called *inscription*, appeared with photographic images. In this phase the machine will go further than the machines of the first layer by inscribing the image.

The third phase will be called *visualisation*: the machines of this kind used in the camera will have as a role to receive the visual object: this means that we cannot visualise the images without the machine, a mechanism of *projection* is needed.
Then comes the phase of *transmission*: the machines allow the direct remote communications with the possibility of multicasting. An example of the machines of this kind is the television.

Real		Beginning of the Period	Machines	Examples
Reproduction of the real	I	Period of Aristotle	Catching	Camera obscura
	II	XVIth	Inscription	Camera
	III	End of XIXth	Visualisation	Movies
	IV	After middle of the XXth	Transmission	Television Video Internet
Design of its real	V	The last quarter of the XXth century	Virtuality	Computer graphics New technologies

Table 1: The different phases of machines

Finally, it is the era of the virtual images, which started a few years ago. The machines here have a specificity that has to be pointed out. Indeed, the computers really change the cycle; they do not behave as the machines of the other four phases where each layer extends the functionalities of the machines of the previous phase. Computers row up towards the source of the network of representation. Indeed, with the image produced by computers, we can say computers generate the *real* itself. In the phase of new technologies, there is no need of all the tools of the previous phases, because they do not reproduce the real, but they create their *own real*.

In fact, the object to be represented is a piece of the computers: the software generates it, it does not exist outside this software. The software creates the objects and models them in its way. The computers are not machines of representation as it was the case for the other machines: they are machines of design. The computers produce their own *reality*, which is also the image. In other words, the two extreme points of the process[8] are joined and connected to form a unique thing. The representation presupposes the existence of a distance between the object and its figuration. With computers, we can say that this presupposition looses its value and meaning, because the fundamental difference between the object and its figuration disappears.

The subjective instances of the device are themselves software's instances: the creator is a programmer, the spectator is a user of the software[9]. Not only there is no *real* now, but also no *representation*. Some critics go further and consider that there are no images. For them even the synthetic picture does not exist: the synthesis is just a set of possibilities anticipated by the program. These critics add that with computers the gaze of the artist disappears no magic and no miracles are produced of virtual world.

b) Similarity vs. Dissimilarity

We will try to tackle in this sub-section the question of realism of the images produced by computers. A central idea that one can found in almost all the machines of images is the mimesis of the reality i.e. the hope of producing the *total image*, an image that is so similar to the reality that it will wholly reproduce real.

According to Philippe Dubois[10], this idea is achieved with the images produced by computers. Indeed from the moment that the machine does not reproduce but generates its own reality, which is an image. It is clear that the question of similarity will have no meaning. He goes further and says that it is no longer the image that mimes the world, it is the real that becomes similar to the image.

---

[8] The object and the image, the source and the result.

[9] One calls that the interactivity.

[10] Dubois Philippe, La ligne générale (des machines à images), *Cinéma et dernières technologies*, De Boek, Bruxelles 1998, pp. 19-39

But, in our case of Visual Assistant, it is really important to have a great similarity of the real. In the conclusion of the chapter *The Visual Assistant project*, we see a remark made by Dominique Serron[11] who stresses the fact that the theatre is based on *impressionism*. If we do not reproduce the real with a maximum of similarity, we make *surrealism*, and not *impressionism*.

c) Materiality vs. Immateriality

This section deals with the question of materiality of images. If we have a glance at a painting, its materiality is directly felt by all the five senses. It is this materiality that makes the painting unique and gives it its artistic value.

In the theatre field, we can also see the materiality of the stage by each of our five senses. That is the materiality of the script, of the stage, of the actors, of the stage objects, etc enables the possibility of making the play unique each time that it is run, by giving its artistic value that it is immaterial.

What about the images produced by computers? The immateriality of the virtual world is extreme. Indeed, we are far from the materiality of the paintings and of the theatre stages. According to Philippe Dubois, the virtual images are more abstractions than images[12]. It is not even a way of seeing things of the mind, it is the product of a complex process of calculation.

He added that the represented movement of an object, a body, just as one sees them on the screen, does not exist effectively in any real picture. The *movement picture* is a kind of fiction that exists just for our eyes and in our brain. Outside that, the *movement picture* does not exist in so far as an object or a matter. That is a brief crossing that gives the illusion just the time of a regard and disappears so quickly that one caught only a glimpse on it.

---

[11] Theatre Director

[12] As normally defined.

2) The real and the virtual stage

The most representative definition of the virtual reality, as it is called, is the following: " *The realisation of a world with physical quality but giving to the spectator, by adequate excitation of all the sensorial system, a feeling with perfect impression to be in interaction with a physical world.* "[13]

The term *feeling* is added to the term *impression* and the first one includes the second. We are concerned with simulating, imitating or recreating the feelings of the reality and, in some way, to come close to the natural perception. One may wonder if the user of a system can have comparable feelings with those lived through a real experience.

If within the Cartesian tradition, it is true that the senses outwit us (sight, hearing, touch, taste and smell), it seems that we are from now on technically able to outwit the senses. It is a form of *trompe l'œil*, enlarged to all our senses.

On the one hand, the stimuli built by the systems trend to reproduce the stimuli felt in the reality. And on the other hand, the type of perception, induced by the device, copies closely the one required by the reality, and tries to include the felt stimuli in the reality with the minimum of shifts and distortions as possible.

The two perspectives do not coincide entirely, and it is not because of the present technical limits of the systems. In fact, we have the character of the simulation, which cannot be exceeded. In other words, we are in the reality without really being in it. The systems invite us to live in an illusion that looks like the reality, but which has just the appearances of the reality.

---

[13] « *La réalisation d'un mode n'ayant aucune réalité physique mais donnant à l'utilisateur, par excitation adéquate de tout son système sensoriel, une sensation avec l'impression parfaite d'être en interaction avec un monde physique.* », Philippe Coiffet, *Mondes imaginaires*, Hermès, Paris 1995.

Why is it so important to have the feeling 'to be in interaction with a physical world'? And, why ask if the generated feelings by computers look like the feelings of the reality? May some unreal feelings exist? Maintaining these questions and differentiation will allow the possibility of comparing of the *real reality* and the *virtual reality* to express the feeling that one can have in front of the possible confusion of the two realities and to point out the dangers.

Let us note that some authors[14] talk about the existence of different layers of reality: *real reality*, *virtual reality*, *augmented reality* and *imaginary reality*. Here one can wonder about the danger of confusing the different realities and even the danger of forgetting about the existence of the real reality.

In all the speeches about the new technologies, which have usually a promotional oration, they speak with a maximum of reductions and a limited conception about what has existed before computer. Moreover, the new technologies practitioners speak of what we are missing if we do not use these technologies, and the technical possibility offered by the new technologies to be elsewhere than where we are in the reality. In other terms, that is the realisation of the ubiquity[15].

With the new technologies, we want to go beyond the human limits, which the technical limits are one of its manifestations. In the *real reality*, we are afflicted by a *lack of feeling*[16] that the simulation systems could fill. Thanks to the new electronic media, our perception could be breaded, the performance of each one of our senses could be improved. We can go further and say that new senses could come to light.

To produce the feeling of the reality, we can say that the virtual stage must be more realistic than the movies and recognised without difficulties by the operator, it has to look like, as much as possible, a stage of the real life. Its interpretation should not have more complexity than the difficulties that we have in the most common ges-

---

[14] Like Philippe Quéau who is a specialist in the virtual domain.

[15] To seem to be everywhere at the same time.

[16] Jean-Marc FICK, *Cybergoût et sensorialité numérique*, in: *Champs Visuels n°5*, L'Harmattan, Paris 1997.

ture of the everyday life. We speak about the gestures that we execute with programmed models, not by the means of a computer. So we need not forget the acquired knowledge. In fact, it is only insofar as we allow the user to insert his/her regular actions in stereotypes scenarios that we can adapt ourselves to the technical imperfections of present systems.

Most of the systems are impressed and dominated by the *obsession of the realism*. In fact, if a system has not enough degree of realism then we do not have the possibility of confusion between the real and the virtual; it cannot be possible to substitute a virtual stage to a real stage. All the means will be used for leaving down the illusion production conditions to the operator.

As the remark in the point about similarity, we can take care the Dominique Serron's remark. In fact, she says that the virtual stage must be perfectly the same as the real stage, without providing a feeling of *artificiality*. She says that the drama is founded on impressionism, and the virtual image gives an impression of surrealism coming from their artificiality.

a) The two different classes of senses: the remote senses and the contact senses

Our body is constituted by a number of different senses: the sense of sight, of hearing, of touch, of taste and of smell. At the base of the virtual reality systems, the general trend consists of the simulation of the senses technically easy to represent artificially. In fact, we have placed in the first sentence the senses, by the degree of difficulty of simulation. Actually, it is easy to represent the sense of sight, but at the opposite, it is really difficult for representing the sense of taste.

With the senses of sight and hearing, as in the plays[17] and in the movies, we have an impression of *presence* and not an *impression* of reality. It is normal, owing to the

[17] It exists some plays without any sounds, i.e. silent plays or well art of mime. It exists one play without the sight sense.

fact that these systems have no technical capacities to simulate other senses than the sight and the hearing.[18]

We can say that we have two classes of senses: the *remote senses* and the *contact senses*. In the first one, we have the senses of *sight*, of *hearing* and of *smell*, and in the second one, we have the senses of *touch* and *taste*.

All researches have studied the synthesis of the remote senses because they are considered more important that the other class. Also, their restitution of the feeling does not require any simulation of a physical contact with the object from where it comes.

We do not discuss more about the senses of sight and of hearing because they are common. But in the case of the smell, we can make the odours with a special chemical device. But, a real problem is to stop the odours instantaneously to have a synchronisation with the other senses like the sight, the hearing or the touch. In fact, we cannot stop them without delay and the odours remain for a long period once made. In brief, we cannot physically simulate this sense now with an immediate synchronisation with the other senses. Of course, in the theatre field, the audience can smell the odours from the stage. But, are these odours in concordance with the represented world in play? In fact, the spectators receive the odours of what it is on the stage, but not the odours of what the director represent in his play.

Among the contact senses is the tactical sense, the more synthesised because it allows the possibility to act directly on the simulated object as in the reality and also because it is the most technically easy to simulate, in regards with the other contact senses. To simulate this sense, the technologies use one or two electronic gloves, which contains a lot of captors that act with the palm of hands. This sense has a real possibility of use in the theatrical field. In fact, the spectators stay in their seat, and it is possible to give them gloves, or different captors to reproduce this sense.

---

[18] H. Rheingold qualifies the movies as *partial virtual reality*, in his book: *La réalité virtuelle*, Dunod, Paris 1993.

Some researchers try to simulate the other contact sense, that is to say the sense of taste, but it is experimental. They use a device that they put in the mouth, just on the tongue. This device simulates chemically the taste. We can think that we will use this sense in theatrical field, maybe with some boundaries. For example, we can imagine a play that it can require the delivery of some edible things to the spectators before the performance, and the spectators are invited to eat the things in some events during the performance.

b) The perfect illusion for misleading the senses

So far so good, we have seen all the senses with no interactions, thinking that they are additional. But, they interact continually in the real life. How to co-ordinate the feeling coming from all the senses? At the start, each sense is isolated, considered by itself, analysed and synthesised, in its autonomous specificity.

The synthesis of each one is more or less complicated, but their resolution is always seen in the same way. It is about to imitate the senses to have a maximum of chance to mislead them. From a scientific point of view, the sensorial processes of the human are badly known at this day. And more, the present applications are imperfect.

To confuse the simulated stage and a stage of the real life, it is essential that the operator does not feel any interference for his/her sight and movements. That is impossible with the devices of nowadays: electronic helmet with visor, electronic gloves, etc. Even when they can adapt themselves on the best of the human senses, we cannot forget their existence.

On the one hand, let us see the case of the sight. To have a real three-dimensional stage, we need two pictures, one for each eye. More, we also may have the feeling of the relief and have the possibility of moving and taking a direction.

On the other hand, to have an impression of discontinuity of the stage, we need a minimum of fourteen pictures per second, otherwise the human eye perceive just a succession of continuous pictures.

Finally, we have also some physical characteristics of the pictures. In fact, a high level of pixels is required so that the picture can make us forget about its materiality. More, it needs enough light, with as many colours as we have in the reality, and with

the information of the texture of the areas and the material of the objects for make us recalling the materials such as stone, wood, cloth, skin, etc. A too big difference could kill the illusion.

The trigger of the action is determined by the combination of visual stereotypes that have to be the most realistic, and by providing the functional information allowing some types of action. Approximately *recreated*, the feelings call to mind the form, the weight, the volume, the texture of an object, and make possible the recognition of this object and its catch in a pre-built scenario. The *real* hand, that is the command organ, sets the virtual hand in action, and the 'virtual' hand returns the feelings that it has in the virtual stage to the *real* hand.[19]

c) The paradoxical situation

A paradoxical situation has to be pointed out: instead of reaching some new unknown worlds, the simulated reality brings the perceived back to the already known. More, it succeeds very imperfectly in that because of the conception of the realism that it brings into operation, and of the present limits of the systems. Thus, we reach a kind of reduction that we schematise in two times:

Firstly, we try to promote the insertion of the operator into the system to adjust this system to a behaviour highly regarded as normal. We continue the automatism linked to the generated practices by the organisation of the social life. We lead the programming of the system towards the reduced conception of the human behaviour[20]. We remove all deviation from the *normality* and if found that we should not do so, a learning is needed; in other words, an adaptation time is required. Of course, that could upset the good training of the interactive system in real time. By reducing the human behaviour to a standard and statistical normality, we produce just an approximate imitation of it.

---

[19] The *real* hand set the virtual hand in motion by the mean of a glove, for example.

[20] In other words, we lead the programming towards the same standard behaviour (normal as it is called) for all the people, coming from the majority of them.

Secondly, we try to separate the humane part of man and all the factors that could lead to think and to transform his behaviour from the feelings that he receives from an external world[21].

d) Feeling and imaginary

The simulation of the direct or indirect contact with an object allows to physiologically release a feeling, but not the imaginary associated to it. Or, all the feelings we have at present time, firstly, reactivate some old feelings and some previous synaesthetic[22] associations that we have already felt or that we have already imagined, and secondly, embed all of that in some new associations and transform them. The question now, will be what about the feelings generated by simulation?

The simulation produces feelings, which have different meanings, depending on whether we try to recreate the elementary feelings (as hot, cold, rough, etc) or to deal with the imaginary that the elementary feelings awake, and by means of which, they will be extended, transformed and deepen. Taking into account the sensitive reactions, that do not have an uniformed well-defined physiological role, implies that the feelings are no longer limited to their strictly informative function and attract the level and the degree of attention needed for the artistic work.

e) Does the virtual world need realism?

We will deal here with another kind of question:

Why does the virtual world aim to be realistic?

The images included in the virtual world have done a *radical and qualitative break-down* with all the different kinds of preceding images. So, why do we want in the virtual world to amplify and to extend some qualities of other kinds of images such as cinematographic images that the specialists of the virtual world declare as outdated?

---

[21] It can be real or simulated.

[22] In other words, the associations between the feelings that we can have at the same time. The associations come from a first feeling, that is the starter.

How can someone understand and justify the split introduced by the computer's images?

In all the systems of imagery, the real pre-exists to the image and it is there that we can find the novelty and the split introduced by the new technologies: the image pre-exists now to the real. As we have said previously, the machines of virtuality design their own real. (See Table 1, p. 6)

Let us have a close look at the field of painting. It is based on the observation of the visible real, and accomplishes some *transformations* to restore this real. Philippe Quéau quotes:

" The classical writing systems try to catch the world with the precision of the layout, the shrewdness of the observation. They try 'to crunch' the real, to determine it with its part of light and shade. "[23]

In fact, the real is painted with a *realistic illusion* coming from the sight of the natural perception. When we see a painting, we recombine the visual real across the picture made by the painter, which is just his/her *interpretation* of the visual real.

Besides that, in the field of movies, we can define the movies as an automatic record of the movement that it recreates visually. For this one, we have the same previous notion like in painting. In fact, when we see the movies, we reconstruct the visual movement of the reality across the filmed movies by the cameraman's style.

In the domain of theatre, we can find some theatre directors that have no closeness with the reality. In fact, some characters can mimic the objects that they are intentionally not present on stage. Moreover, some drama works are surrealist, and the spectators must interpret themselves the play.

---

[23] « *Les systèmes d'écriture classiques cherchent à saisir le monde dans la précision du tracé, la finesse de l'observation. Ils cherchent à croquer le réel, à le cerner avec sa part d'ombre et de lumière.* », quoted by Philippe Quéau.

Whether the real pre-exists or not to the image, the representation of the real cannot be thought outside the interactions between the person who composes the image and the object, real or not, to which the composition is applied.

The interaction varies with the habits of perception proper to each historical period. Even, for the same period, the interaction varies according to the *pre-built* representations. These pre-built representations are strongly related to the education and the culture which modify and program the sight one can have of the world.

We can say that the split introduced by computer science is not really a novelty. It is just more apparent than in other fields.

If we know that in every artistic domain[24], the *perfect realism* is never an objective in itself then we remain with no answer about our question on the realism that people are expecting from the virtual world.

f) Is the virtual a substitute of the real?

Is it not abusive when some people talk about the substitution of the virtual to the real?

If we know that the artistic approach consists of making things visible and not representing the visible, we can then say that it is excessive to speak about the substitution of the virtual to the real. It is just another kind of sight that someone is having inside the real world besides that a category of virtual consists on restoring the sensible appearance of the real.

What do the artists '*reproach*' to the virtual?

It seems that what artists '*reproach*' to the virtual world is the absence of artistic *intentionality* and they argue that digitising an object consists among other things to choose some strokes and excluding some others. By doing that, there is, according to them, a kind of reduction of the set of possible. That is a lack of realism attributed to

[24] Painting and movies are only examples.

the virtual world, or the lack of realism is the essence of the artistic approach. In fact, the artistic gesture consists precisely in operating some choices as far as the image is concerned.

The theatre director, when his/her script is based on an existent textbook, does in some way a choice of what (s)he must take, and what (s)he must leave, what to emphasise and what to tarnish.[25]

As a consequence, this argument cannot be accepted and we have to explore other sides to prove the absence of artistic *intentionality* within the virtual world. The only thing that we can confirm at this level is that while artists and the specialists of the virtual world are doing the same thing about the images, i.e. making choices, there is a difference between them in doing that. Indeed the choices performed by the artist appear as a kind of innovation[26], which is missing in the images provided by the virtual world.

But, nowadays, the choices performed in the virtual world are done in function of some constraints coming from a certain conception of the realism and not in function of a project of rebuilding the world from some artistic options. These artistic options are the only ones able to create some new original worlds independently of the technical tools and means used. The Art has never waited for whatever *perfect realism* for existing as Art. In the artistic domain, the relationship with perfect realism was never an objective in itself and is only achieved by a re-creation of the real.[27]

---

[25] But, Dominique Serron does not agree with this method. She prefers have the whole of the drama work. (See chapter: *What is Theatre Direction*)

[26] This innovation could be the expression of his experience, his dreams or his imaginary.

[27] Gérard Leblanc, Quelle autre scène ? (Réel/Virtuel), *Cinéma et dernières technologies*, De Boeck, Bruxelles 1998, p. 64

g) The two kinds of virtual reality: the duplication and the new universe

Confusion is maintained between two modes of solicitation of senses: the first is in the everyday life and the second is in the artistic practice. In the artistic practice, the senses are related to what one can do on the imaginary. The systems of simulation revive an old question in the domain of the Art: "*The interaction between Art and Life*"[28].

But, we have to remove the confusion that exists between the two modes of solicitations of senses. Besides that the virtual has to tie up with a philosophical tradition that considers it, as said by Aristotle: « puissance en acte », as a component of the real included in the present time. These two modes have achieved to enable us to consider the virtual as a technique that has appropriated a concept older than itself, i.e. the concept of the interaction between Art and Life.

In this way, the virtual could be seen as an opened direction on some original forms of life. The forms of life are achieved by a certain kind of materialisation of the imaginary.

> "*The recreation of the feelings of the reality allows the existence of what does not exist.*"[29]

The recreation of feelings has to deal with two different types of things. Firstly, it can bring into existence, under other forms, things that already exist. Secondly, it can bring into existence things that are only in our imaginary.

---

[28] Gérard Leblanc, Quelle autre scène ? (Réel/Virtuel), *Cinéma et dernières technologies*, De Boeck, Bruxelles 1998.

[29] Ibid.

Depending on the subject of the recreation of feelings, i.e. something that has existed or something belonging to the imaginary, we can have two different kinds of virtuality:

1. The first is in the order of *duplication*. The simulated reality substitutes the reality by redoubling it. It becomes integrated necessarily in the previously constructed representations. It will just (re)produce stereotypes derived from the reality.
2. The second is able to allow the existence of more than one universe that have ever been lived. *The presuppositions are radically different* from those of the first kind of virtuality. With this second type of virtuality, we could perceive some other things and we could do things, which we do not have in the *normalised* life, which is conducted by the social organisation of day-to-day life.

Allowing the interaction between the contact senses will enable the creation of some worlds that has never existed. Every artistic project tries to create such worlds. The systems of simulated reality could allow some of these worlds. The condition to reach it, is that not only the artists could use the existent software but also they can contribute in the elaboration of new ones by integrating some new dimensions, which could reduce the gap, and help to take into account the specificities and requirement of their field.

3) Conclusion

This chapter is for this work as a light, something that enables the programmer that will design a program for an artistic purpose, to understand first the difficulties and the limitation of what he is attempting to do. Second to understand and to explore the questions that the artists are rising about the use of computers in their domains.

This chapter can be seen like a *bridge* that enables to fill the gap between the two participating parts, i.e. the *programmer* and the *artist*. Especially to allow the programmer to understand the boundaries of what is possible to do with the virtual reality and what the artist reproach to the virtual reality.

C The Visual Assistant project

We describe in this chapter the Visual Assistant software and also the current development. The majority of this section is derived from Colin Beardon's research[1]. we have also used the previous dissertation of Frédéric Miche[2]. We try to defend the use of this software, especially in the academic theatre production.

It is really important to justify the need of this software, and particularly in the field where we want it to be used. Of course, some other fields are likely to use it also, but that is not the issue here.

We will try to find out how it can help the work of the theatre production, and we will try to compare the Visual Assistant to a studio. In fact, we want to prove that

---

[1] Especially the Visual Assistant website at: http://www.esad.plym.ac.uk/projects/VA.html
and Colin Beardon, Computers and Improvisation: Using the Visual Assistant for Teaching, *Digital Creativity*, 1999, vol. 10, n° 3, pp. 153-166

[2] Frédéric Miche, Chapter Four, Visual Assistant: A support for creative practices for theatre, *Computer Sketching: How Software Tools Can Enhance Human creativity?*, Institute of Computer Science (FUNDP), Namur 1999.

the software is useful and that it can change all the work of the teachers and the students. We are convinced that it can improve the visual creativity.

At the end, we explain the characteristics of Visual Assistant. The use of the different components of the software is explained by the means of a practical user guide that we added in the end of this chapter.

1) Introduction

The Visual Assistant is a software designed and developed in a research program to explore the possibility of creative advantages with digital technologies. The theatre is the field of this investigation and we try to see the potentiality of those technologies in enabling the production artists to better express, communicate and develop their visual and spatial ideas.

In the environment of creative people, the ideas about what the computer could play a role in bringing a wonderful aspiration may seem surprising.

> " *Traditionally, computers have been thought as being opposed to creativity, as requiring great amounts of training, as being demanding of detail, and as lacking any spontaneity.* "[3]

The Visual Assistant project aims to produce computer software that requires little expertise, yet can allow anyone to express visual and spatial ideas relevant to a performance.

In the academic field, the Visual Assistant aims to offer a new and creative approach towards this understanding by providing a flexible environment where a student can be designer, director, actor and audience at the same time.

Of course, some softwares already exist for such purpose in the form of 2D image processing or 3D model building and multimedia applications. But, all of these softwares cannot do what the artists in the field of the theatre want.

---

[3] Colin Beardon, Computers and Improvisation: Using the Visual Assistant for Teaching, p. 153

2) Computer and creativity

We have investigated this notion in a previous chapter. We have shown that the notion of creativity is open to debate. We have said that the creativity is not easy to define, and we have tried to understand this strange feeling by exploring authors' opinions, and philosophers' ideas about the subject.

For the purpose of our work,

> *"... creativity is a mode of interaction with the world in which the systematic and routine are not fully formalised, thus providing a space within which it is possible to explore and to imagine."*[4]

All disciplines, all human practices, involve a degree of such creativity and in all practices it seems inevitable that comes a later stage at which a more formal mode of interaction is needed in order for a specific outcome to be materialised.

To date, computer technology has been mainly associated with less creative and more systematic mode of working. Computers have typically been used for the detailed construction and transformation of ideas and they have become useful for producing a final clean product. Many artists and designers think of the computer as a means of copying with the detailed and technical aspects of their creative work and many would see it as inherently opposed to the processes of *true* creativity. That opposition may have historical reasons, but it is the core attitude of the development of the Visual Assistant Software.

---

[4] Colin Beardon, Digital Creativity in the 21st Century, p. 4

The Visual Assistant project aims to address the following research question:

> *" Is it possible for us to use computer based tool during the creative phase of work to real advantage? "*[5]

It does not assume a positive answer, but rather, seeks through new software design and interaction of this into creative practices, to elicit a clear positive answer.

3) Design of Visual Assistant software

Meaningful and useful software needs to be grounded in the practical concepts of its intended users and provide a code within which interesting designs and artefacts can be produced. But, all of these softwares cannot do what the artists in the field of the theatre want[6]:

- *Sketching:*
  Sketching stage of a design is necessary to work with ill defined ideas and all existing software require too much detail.
- *2D and 3D representations:*
  2D graphics does not enable the proper exploration of three-dimensional space, whereas 3D modelling requires technical skills that are not easily achieved.
- *Look and feel:*
  None of the interfaces look and feel like a theatre, they look and feel like a computer interface.

---

[5] Art. Cit., Colin Beardon, Digital Creativity in the 21st Century, in: *Proceedings of ICFAD '99*, p.4

[6] Colin Beardon and Terry Enright, The Visual Assistant: designing software to support creativity, in: *Proceedings of CADE'99 Conference*, University of Teesside 1999, p. 1

Visual Assistant software provides an imaginary space into which two-dimensional images can be placed and manipulating in three dimensions, in order to quickly create visualisations relevant to performance. The Visual Assistant software is based around three concepts of representational forms[7]:

- *Sketching:*

  Sketching is taken as a mode of expression in which detail is less important. There is also certain plasticity about the sketch itself, in that it is a disposable medium. It is not intended to be permanent but rather a quick attempt to capture something. If it fails one can dispose of the product without regret and try again.

- *2D and 3D representations:*

  The computer screen represents 2D objects quite well, and it can simulate $2\frac{1}{2}D$[8] through the layering of images, but 3D representation on a flat computer screen is never intuitive. We struggle with pitch and roll versus pan and zoom; there are too many things to do within a flat surface.

- *Collage*[9]*:*

  Computer graphics has extended the tyranny of the fixed-point perspective. For computers, geometry provides the ideal abstraction for representing space in two dimensions. Artists and designers have moved on. Through collage, for example, it has been possible to avoid closure and to allow the viewer to contribute to meaning. It provides a natural medium for deconstructing messages and placing them in new contexts where they convey new meanings. It employs a combination of borrowing and re-presenting which is familiar to many in the theatre world.

---

[7] Art. Cit., Colin Beardon, Digital Creativity in the 21st Century, p. 5

[8] We can use the sight illusion to represent the 3D objects on a 2D plan. By the means of the perspective, we match the 3D objects to their correspondent 2D objects, which are on the 2D plan.

[9] Arts: a picture made by sticking other pictures, photographs, cloth, etc onto a surface.

4) Goals of Visual Assistant

The Visual Assistant has a clear context of use. Of course, there is no guarantee that it will be used in the way we intend[10], but the design approach anticipates this phenomenon. It concerns a number of people involved in the real production of play, each with a distinct role and with a great need for co-ordination and co-operation. Usually, few people are likely to be involved in the visual realisation, but all have ready access to the play text. Could there be a role for digital technology in providing a common canvas for the visual and spatial aspects of a production? What could such software be like?

The software was designed for users who are primarily interested in the theatre and are probably not at all interested in computers. We can see the Colin Beardon's initial design objectives for the software[11]. These aims seem to be drawn from his experiences of working with theatre students.

- *It must be very simple to learn:*
  someone should be able to see someone else using it and then confidently use it himself/herself. The principle interest of the users is theatre, not computers. They do not want a manual or a tutorial to read.
- *It must be simple to use:*
  there is little point in providing 96 options when most users will only use 12. More, as the previous point, the theatre, not computers, interests the users..
- *It should give meaningful results quickly:*
  there is a tightly iterative process of producing and evaluating.
- *It should use visual imagery and locate it spatially:*
  constructive geometry is insufficient, for example, without texture mapping, but we have to keep it simple.

---

[10] Colin Beardon, Sue Gollifer, Christopher Rose, Suzette Worden, Computer Use by Artists & Designers, *Computers and Design in Context*, in: *Proceedings of MIT Press*, Cambridge (Mass.) 1997, pp. 27-50

[11] Art. Cit., Colin Beardon, Digital Creativity in the 21st Century, p. 5
Art. Cit., Colin Beardon, The Visual Assistant: designing software to support creativity, p. 2

- *Products should be disposable:*
  we do not want so much resources spent on a computer model that the user sees the model as more important than the real world. It should not matter if the product is transformed or destroyed.
- *It must be a believable world:*
  the screen should look like a theatre, not a computer. When acting on the computer screen we should be thinking as a theatre person.
- *What happens should be like sketching:*
  it should not matter if work is irretrievably transformed or destroyed and user can always try again. The Visual Assistant should rather be seen as a theatre image-making sketchbook rather than as a theatre design tool. In fact, the Visual Assistant has been developed as a sketching tool to offer artists and designers tools that can be used during the initial stages of the creation process, which does not need details before its final stages.
- *It should support* process *rather than* product*:*
  particularly processes that lead to clearer understanding and better actions in the real world, e.g. a better perception of a play to put on.
- *It should present a believable 'language-game':*
  when acting on the computer screen user should be thinking as a theatre person. That is, the software tool should move aside, hide behind the theatre practitioner. It should become free and easy for its user rather than constrain him/her to submit to its sequence of proceedings.
- *It should support person-to-person dialogue:*
  the main role of the output is to serve as a common sketchpad to support critical discussions. Anyone can express an idea and anyone else can explore and talk about it.
- *It should be able to lead to a more detailed model:*
  ideally, if a successful outcome arises, it should be possible to go from what has been produced here to software designed to add greater precision.

Colin Beardon, The design of software to support creative practice, in: *Proceedings of IDATER 1999,* Loughborough University 1999, p. 5

5) Theatre Director's Requirements [12]

Visual Assistant is not only intended for students, but for most of people involved in a play. As a consequence, we propose to complete the goals of Visual Assistant as stated in the previous section, by getting the requirements of theatre directors. In this way, we can have another investigation besides the objectives from the Colin Beardon's works. In fact, Colin Beardon has seen the requirements in theatre education but not those coming from the professional theatrical field.

The major expectation of Dominique Serron from computer science would be the ability to calculate the contact between audience and actors. That is to say, the computer would calculate the distance between the audience's position and the stage's organisation: how to place the audience, at which distance should they be from the stage, how much distance should there be between one spectator and the other, in which type of seat, position[13]...

Not only should software permit to calculate that information, it should also be able to simulate the different feasible placings.

The aim of such an approach is to discover the emotions certain placings provide, because the emotions the direction wants to arouse do depend on the spatial arrangements of all stage elements such as the audience, the actors, the stage and the scenery.

The theatre director would like to be in the audience's place in order to make out the emotions that are generated in it. Doing so, (s)he wants to succeed in transmitting it his/her own interpretation of the play.

On a more specific domain of scenecraft, it would be interesting to view costumes, scenery and particularly the colour's intensity, the latter being the most important. Visualising the costumes and the different type of cloths existing on the screen could be a tool. Moreover, every costume consists of various cloths, which can be

---

[12] A conversation we had with Dominique Serron, theatre director.

[13] That is to say standing, sitting, lying, etc.

combined to see what matches or clashes. The possibility of modifying the cloth's colours can also be a help, colours being determinant to make cloths assemble or not.

Concerning scenery, a library of themes and objects combinable, in order to provide a certain effect, could base its conception. Moreover, the stage dimension and its depth must be modifiable to think up the space arrangement.

The colour issue is very important. Actually, it would be marvellous to assemble all the elements to get a general impression of the colours. An example: suppose that we choose to dress a character of little importance to the stage action in black, according to his/her tiny role. Then the other actors' colours, the projector's intensity, the scenery painting might still cause the one we wanted to be secondary, to become the major visual attraction to the audience. Then simulation will give us a preview of that undesirable effect.

The computer access must be as simple as possible, its manipulation easy and comprehensible for an artist, and all habits of the computer world avoided so it is adapted better to the theatre branch. It would be great to talk to the computer: as a director's job demands the giving of instructions and the coordination of the different elements by his/her voice, a vocal characteristic would be just out standing. On the other hand, the mouse, keyboards and screen are convenient, if their handling is explained.

Prints also have to be an essential element of the software. They would enable us to fix the ideas for the stage arrangements or to compare different arrangements. We could even show them to other persons and talk about them.

At last, it would be useful to have a three-dimensional view of the theatre hall[14]: the stage, lighting, audience placement, ... a virtual helmet would permit to visualise and to talk around in this virtual reality[15]. To build oneself an approximate image of the placement, lightning and distance effects... to obtain a sketching out of the production. And, why not filming this virtual reality to save it on a videotape to visualise it for many and to discuss about?

---

[14] Through that is maybe a matter for science fiction or the experimental domain. (See appendix *French Translator*)

[15] See chapter: *What is the relationship between real reality and virtual reality?*

Visual Assistant replies in some needs: it enables the scenecraft, the design of scenery. Next, the colour management is possible with Visual Assistant. It permits to assemble all the elements to get a general impression of the colours.

Furthermore, Visual Assistant allows the possibility to catch pictures from libraries coming from CD-ROM or even Internet[16]. In this way, we can try the combination of different objects, in order to try the different effects. Moreover, we can change the stage dimension and its depth to think up the space arrangement.[17]

Its manipulation is easy and comprehensible for an artist, and it tries to put out all habits of the computer world in the way to provide the computer science adapted to the theatre branch. The mouse, keyboards and screen are the input-output system. We think that is an easy system to learn.

Finally, we can save the work made with Visual Assistant on discs. Furthermore, we can save this work as pictures that we can print with paint software. It enables us to fix the ideas for the stage arrangements or to compare different arrangements with printed pictures.

---

[16] See point about *on-line archives* in *state-of-the-art*

[17] Just in version Macintosh at this time.

6) Computer vs. Theatre Studio

Let us quote Spolin's book:

> " *If the environment permits it, anyone can learn whatever* [he/]*she chooses to learn; and if the individual permits it, the environment will teach* [him/]*her everything it has to teach. 'Talent' or the 'Lack of talent' have little to do with it.* "[18]

We can interpret the Spolin's remark in the field of this work and see that the environment of the computer can be analogous to that of the theatre studio thus enabling students to use the same techniques of improvisation in virtual as they do in actual space. The difference being that the emphasis in virtual space is *visual* while in actual space it is *physical*, including voice. Educationally the combination and juxtaposition of these two learning environments can significantly aid the educative process for theatre students by emphasising the physical alongside the visual.

The theatre practitioners are those who can perceive and conceive performance from all perspectives, but most of the time, visual is the less represented area in artistic schools.

The computer, like the studio, can become a site of play where the emphasis is on eradicating blockages, challenging stereotypes, being receptive to the unexpected, and the unusual. Some of the formal characteristics of play, as the creativity or the improvisation, can be seen in the studio or on the computer.

---

[18] Spolin V., *Improvisation for the theatre*, Pitman Publishing, London 1963.

As said by Huizinger:

> *"... it is a free activity, standing quite consciously outside 'ordinary life' as being 'not serious', but at the same time absorbing the player intensely and utterly ... It proceeds within its own proper boundaries of time and space according to fixed rules and an orderly manner."*[19]

What about the graphical interface? Visual Assistant software seems to be a user-friendly environment that has the elements needed to promote a sense of creative freedom. The remaining technicalities are comparable to the techniques, conventions and rules that apply in the studio. The more we play the more we become familiar with the software, and the more rules we create the greater becomes our knowledge of the environment which is generating the playing. There is a tension that occurs in all the creative processes between spontaneity and discipline.

We think that the Visual Assistant software is a powerful tool and environment to develop and promote visual creativity. Its application in the study of theatre promises a breakthrough for it is both a complementary site for the training of visual perception, and a dynamic sketching and storage for pictures.

---

[19] Huizinger J., *Homo Ludens*, Beacon Press, New-York 1955.

7) Software development method

Visual Assistant software is primarily for people working in the theatre and concerned with performance, but it may have many other fields of application. It is an essay to see the limits of representations in two dimensions or even in three dimensions. The software allows the users to import, to draw, to modify 2D graphical objects and to manipulate these objects within 3D space.

To design the software, we can examine uses of language in the artistic field, and in this way, we can understand working practices and design better products. But, the problem in the case of art and design is that is not primarily text or speech based.

Moreover, the meaning of the software must not be defined by the software designer or the computer or the end user, but it is negotiated by their collaboration. The partnership is really important in the fields where we have just an abstract things to think the software. In our case, we want to make software for creativity which is an abstract notion. We can define the creativity as the search for meaning or for the expression of meaning. But, the meaning therefore cannot pre-exist the practice[20]. The more we involve the user in the design process, the better software will emerge. In fact, users can say directly what they need and may help to discover some new ideas and mistakes.

The biggest problem, we have in our case, is the usability of creative practitioners to test a field of work which is very different from their experience. In fact, the computers are only used to perform clearly defined tasks; typically to transfer information unambiguously from sender to recipient with the minimum of information loss. In this way, there is no recognition that the computer might be used in conjunction with imagination, or even with play. More the computers are used primarily to convey information encoded as letters and numbers within a limited two-dimensional plane. In that way, there is no recognition that multimedia computing generates subtleties of meaning which can be harnessed to great effect.

---

[20] We have tried to explain this notion in the chapter about creativity.

We can see the preliminary investigations of Colin Beardon to explain the misunderstanding between artists and designers, which are for him unlike users of software:

> *" They will often say that they only got interesting results from software once something went wrong. Some elevate this to a technique, forcing the software to misbehave or deliberately subverting the intentions of the software designer in order to get an interesting response. "*[21]

Through this anecdotal remark, he said that the artists could not understand the problems of software designers. In fact, they cannot understand the process *generate and test* used by the computer field:

> *" I am surprised just how many times I have heard the chance remark that the progress was only made when the software did something unexpected. "*[22]

The designers refuse to engage in a constructive criticism of the tools and creative users deliberately try to subvert the intentions of software designer. Knowing that, it is really difficult to make software implying the creative practitioners.

From another point of view, the major organisational problem has been the software and the computing equipment. The initial decision was token to develop the software for the Apple Macintosh; but it created a problem when many users switched to PC platforms.

---

[21] Art. Cit., Colin Beardon, The design of software to support creative practice, p. 3

[22] Ibid., p. 3

8) Visual Assistant in academic theatre production

The Visual Assistant software has been used in several teaching workshops with purely educational purposes. It offers many advantages in this field:

- Ease-of-design;
- Rapid prototyping: sketching;
- Re-education theatre students;
- And, last but not least, cheaper than setting theatre properties on an actual stage.

Its use in academic theatre production could help to educate the students on the right track.

In theatre production, students have to learn a lot of methods: how to design stage sets, how to move the actors on the stage, how to put the objets on the stage, how to make the shifting of the actors and the objects, etc. Of course, this is a rather costly activity, both in money and time. Each student should need a real theatre stage; a physical theatre for a whole classroom, one student at a time, one stage for each student at a time.

Moreover, it also means that we should have real objects like furniture. It is time consuming to place all the objects, and there is a need of a warehouse for stocking all these objects. And, if a student would choose an object by mistake, and after (s)he has try it on the stage, (s)he says that it is not the good object: should we buy another one if it is not in our stock? We can consider that the change of position of the object can improve the time of arrangement. In fact, some of them are very heavy, and we should have some people just for managing the stock, the order of objects, and the objects moving.

Finally, what about people or dummies for the students to figure out where to place everything and which space they need on the stage? It is realisable to book a lot of people who do a lot of things out of our activity. And the most important, labour is very expensive. Evidently, this is not realist and feasible.

As a result, an objective of the Visual Assistant software is to save time and money by allowing the students to use a computer to decorate a stage, to move the actors on a stage, to place the object on a stage... In this way, the students have a preliminary idea of what they want. Of course, they can do their stage in life-sized if it is necessary.

Now we have access to the television and movies from early childhood that many students conceive many aspects of the theatre in terms of television or movie techniques. The close-up desire is probably the best example of this observation. In fact, the close-up has no place in the theatre field. Further, they also think about theatre only in terms of *words*: the words of a play text, the words of critics and theoreticians, the word written by students and the words presented by lecturers. Theatre is more about doing, it is about action and space and light and many other things. If computers are to be used in this field of creative practice to real advantage, then there is a real need for a range of visualisation software designed for theatre practitioners to help address this imbalance. To correct their visual and spatial skills, they need to develop their skill of stage designers by the use of real physical stage. But, as said before, it is not realisable because it is often impracticable and expensive. But, the Visual Assistant software offers such potentialities since it represents the stage as a whole, using 2D pictures as theatre properties or actors and the possibility to explore spatiality by setting out objects in a 3D virtual space. The Visual Assistant software does provide an accessible and fairly quick means of getting students to think visually about a play, not just in terms of particular pictures to be included but also in terms of colour, mood, atmosphere... By providing a sketching environment, many ideas can be explored, discussed and amended.

9) Overview of the software

As said before, 3D modelling softwares produce accurate 3D models of theatre layout and stage and set design such an approach assumes a knowledge of detail and a spending time that may not be available or even desirable when ideas are in formation. The Visual Assistant provides a quick way of creating the general impression generated by a stage, or other 3D space, so that the important concepts are clarified and can be discussed.

> *" The environment has a certain logic, but it is not based upon photo-realistic geometry. The uncompromising lack of photo-realism has been found by educators not to be a weakness but a strength of the software. It forces student users to express ideas approximately, yet in a way that creates what one practitioner has described as 'atmosphere'. "*[23]

The Visual Assistant software enables the user to collect textual and graphical material and to organise it visually. It builds upon the visual and spatial qualities of objects on the screen, rather than their formal (any semantic information such as object type, name, etc.), physical (location, size, image) or textual properties (any written comments). It enables the user to construct abstract or quasi-realistic representations – the latter ones could actually become real stage sets for real theatre plays.

The software locates visualisations within three dimensions and the user is presented by a space. There are two-fixed viewpoints: from 600 units in front of the stage[24], and one from directly above the stage looking down.

---

[23] Art. Cit., Colin Beardon, The design of software to support creative practice, p. 4

[24] See chapter: *Software Architecture*

Let us take a look to the software's interface: (See Figure 1)

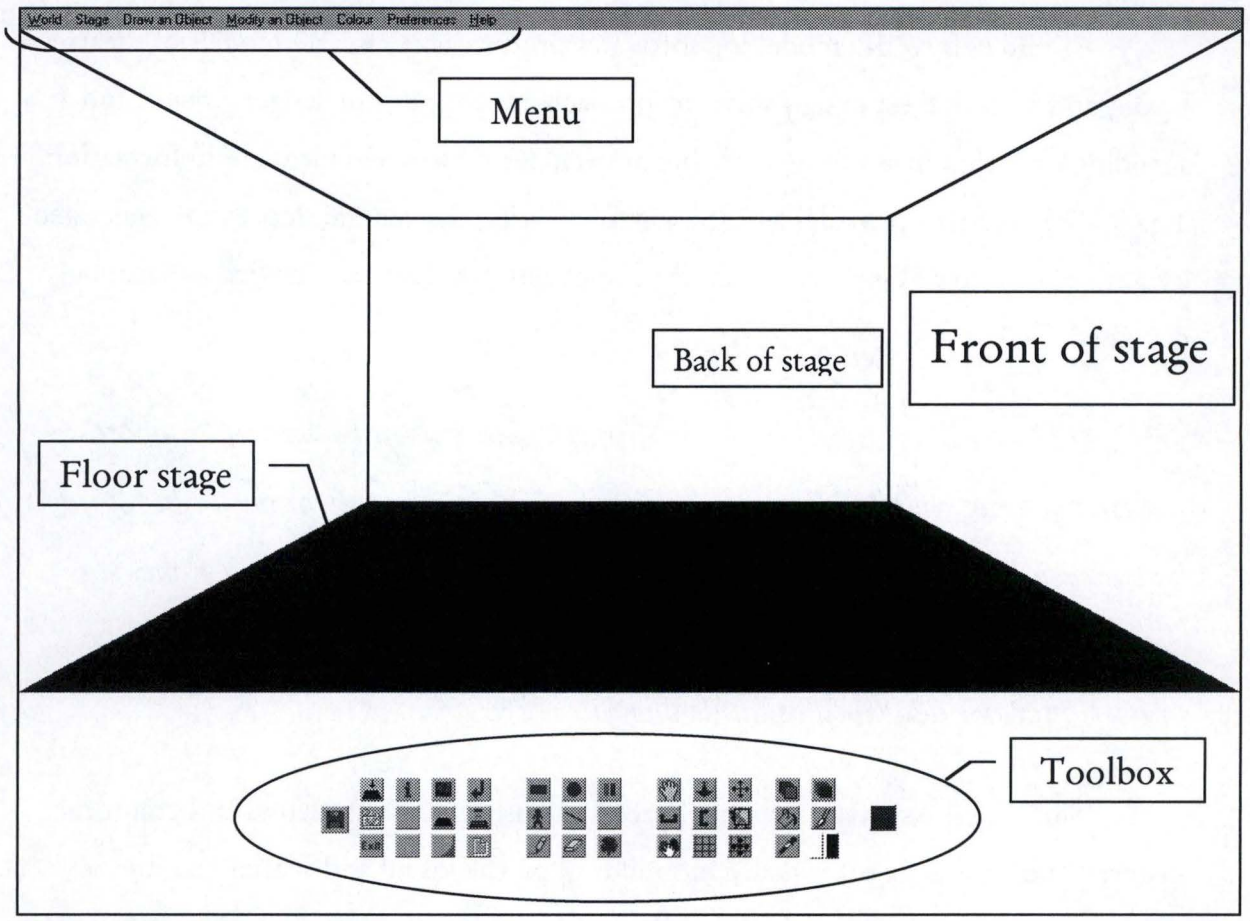


Figure 1: ***Visual Assistant's interface*** **(in Front View)**

The fundamental architecture of the Visual Assistant software is that it contains a sequence of stages, each of which contains a number of objects[25]. Objects can be imported graphic images, one of a number of predefined constructs (circle, rectangle, and line) or a pencil drawn object. Once entered the image can be manipulated by changing its size, flipping it, erasing portions of it, recolouring it, turning it into a pattern...

Each stage is bounded by a three-dimensional box in perspective 2 $\frac{1}{2}$ into which objects can be placed. All objects in the stage have a visual two-dimensional representation. When viewed from above the various images appear as lines with width but minimal depth.

---

[25] Called *StageObject* in chapter: *Software Architecture*

They can be moved around in this top view and the moves are permanent, i.e. they take effect if the user changes back to front view. By moving the ends of objects in top view, objects can be rotated around their centre point (i.e. they do not need to face the front of the stage). Back to the front view, the moved objects appear greater or smaller if they have been moved forward or backward respectively. Objects rotated in the top view appear like a trapezoid in the front view. In fact, this simulates a perspective effect due to a leaned object.

In the front view, one can see an outlined rectangle, which represents the back of the stage, with four lines linking its corners with the corners of the display area. Together, they represent a *box* bounding the stage. This *box*, called the *Wireframe*, can be switched on/off if a designed stage requires no spatial limits.

With the software, we can make a movie with thirty stages in a sequence. There is an option *to play* the sequence of stages and this displays a simplified animation, rather like a slide show.[26]

[26] At this time, this characteristic is not implemented in the PC version. But built architecture of PC version takes care about this future functionality.

10) Table of functionalities

The below tables show all functionalities implemented in the software[27]. We have put the two columns for the versions: one for the PC version[28], and another one for the Mac Version. We can see that the PC version has less options implemented because it is a prototype version, and the development is not totally finished, but all basic functionalities are implemented, and we can load and save our works. In this manner, we can give our attempts to some friends to discuss about the work to improve it.

- *World* Menu

First, we have the general functionalities that allow loading and saving of Visual Assistant works, i.e. plays or scenes, with the Visual Assistant Software. Some functionalities have three dots "..." to show that this process executes the displaying of a new window, on the main window, to catch information from user.
(See Figure 2 and Table 1, which is on the next page)

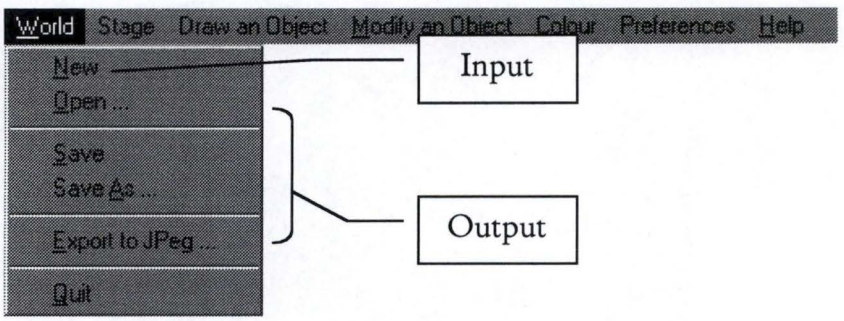


Figure 2: *World* Menu

[27] These tables explain the functionalities of the two versions that we can find on the CD-ROM provided with this work.

[28] In this column, a cross 'X' represents the functionalities implemented by myself and a 'C' the functionalities implemented by Colin Beardon.

Table 1: functionalities to save or to load the play (Menu *World*)

Tool + = double-click on the tool	*Item*	*Description*	*Mac (v.1.5)*	*PC (v.0.5)*
	New	Will create a new, empty environment. If the current environment has been altered since the last New, Load or Save command then users will be asked if they wish to save the present environment.	X	X
	Open ...	Will load a saved environment. If the current environment has been altered since the last New, Load or Save command then users will be asked if they wish to save the present environment. Users will be asked to identify the file they wish to load (usually called ".va"). If successfully identified, the environment will be loaded.	X	X
	Save	Will save the current environment with the same filename as the last Save or Load command. The saved file will contain all the images used in the environment. The previous version of the file will be automatically replaced. If no previous Save or Load command has been issued, the effect will be the same as 'Save as ...'. This command is only available if the environment has been altered since the last New, Load or Save command.	X	X
 +	Save As ...	Will prompt users for a folder in which to create a new folder to save the current environment. If a folder already exists with that filename, users will asked to provide another name. The environment will be saved as a text file in VRML 2.0 format '.wrl' along with '.gif' or '.jpeg' files for all associated images. This makes it readable in any VRML browser.	X	X
	Export to JPEG ...	Saves a flat 2D image of the stage as a JPEG file.	X	
	Quit	Will quit the Visual Assistant and return to the desktop. If the current environment has been altered since the last New, Load or Save command then users will be asked if they wish to save the present environment before quitting.	X	X

- *Stage* Menu

Second, we have the functionalities that allow the management of the different stages. These functionalities are applicable on all the current stage that we see on the screen. (See Figure 3 and Table 2)

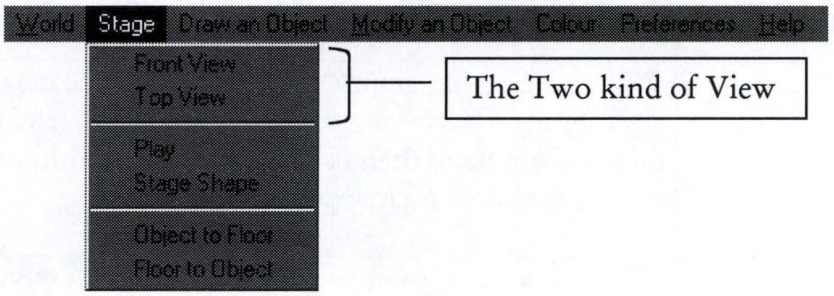


Figure 3: *Stage* Menu

Table 2: functionalities about Stage (Menu *Stage*)

Tool + = double-click on the tool	*Item*	*Description*	*Mac* (v.1.5)	*PC (v.0.5)*
	Front View	Changes the viewpoint to a position directly facing the centre of the back wall and at approximately one stage depth from the front of the stage.	X	X
	Top view	This command permits viewing from a point directly on top of the stage. Objects will appear as thick black lines. When selected with the cursor, a full size image will appear and this may be dragged and dropped anywhere within the stage limits. The effect of changes made in Top View will be permanent and will be reflected when you return to Front View.	X	X
	Play	This will play through the 20 stage settings in sequences. It is a simple animated sequence with some in-between. The toolbox and menu bar will be hidden for the playback. Use the space bar to interrupt the playback if desired.	X	
	Stage Shape	This command permits to change the kind of stage: trapezoidal, circular, square, ...		

Tool + = double-click on the tool	*Item*	*Description*	*Mac (v.1.5)*	*PC (v.0.5)*
	Object to Floor	Click on an object and it will be removed from the scene and pasted to the floor (and shown in perspective). Once on the floor it can be flipped, patterned, etc. and removed from the floor by using the 'Floor to Object' tool.	X	C
	Floor to Object	Click on the floor: if it contains an object then that object will be restored as an ordinary object in the scene (at the same location, size, etc. as it occupied before it was pasted to the floor). The floor will resume a plain colour.	X	C

- *Draw an Object* Menu

Third we have the functionalities that allow drawing the standards shapes. More, we have the *Import...* functionality that enables to take pictures from files. (See Figure 4 and Table 3, which is on the next page)

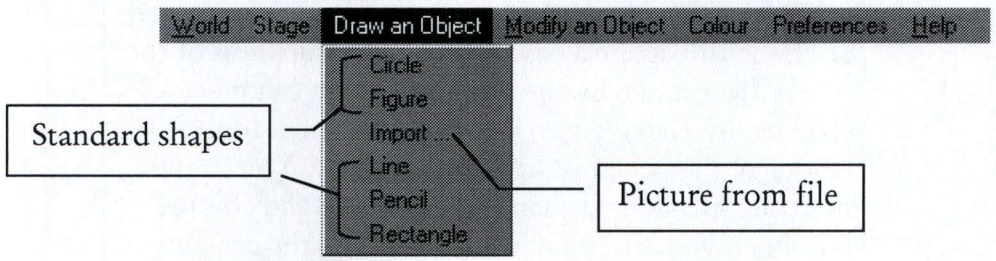


Figure 4: *Draw An Object* Menu

Table 3: Functionalities to draw new Stage Objects (Menu *Draw an Object*)

Tool + = double-click on the tool	*Item*	*Description*	*Mac (v.1.5)*	*PC (v.0.5)*
	Circle	Create a new circle at bottom – front of stage.	X	X
	Figure	Create a new figure at bottom – front of stage.	X	X
	Import ...	You will be asked to select a PICT, GIF or JPEG file. A picture, in the same proportions as the original image, but reduced if necessary to fit the screen, will appear centred on the screen (located at the front of the stage). The image of the object will be shown as 'transparent' (i.e. all white pixels will be transparent). All images are represented internally using thousands of colours at 72 dpi.	X	X
	Line	Create a straight line in the current colour. The width of the line is determined by pen size. (One way to change this is to double-click on the line tool.)	X	X
	Pencil	Enables you to draw with a pencil. If you start drawing on an existing object (at the front of the stage and non-angled), then your drawing will add to that object, otherwise it will create a new object. The thickness of the line is determined by the line size (which can be changed by clicking two times on the pencil tool). The colour of the pencil is the current colour. You can use multiple strokes and change the pen size and colour, but when you select any other tool then the drawing will stop. A new drawn object scan be moved, resized, coloured, flipped, erased, etc. and grouped, just like any other object.	X	X
	Rectangle	Create a new rectangle at the centre – front of the stage.	X	X

- *Modify an Object*

Fourth we have the functionalities that allow to modify the characteristics of drawn objects on the stage. Moreover, we have *Undo* function that enables to cancel the last operation. (See Figure 5 and Table 4)

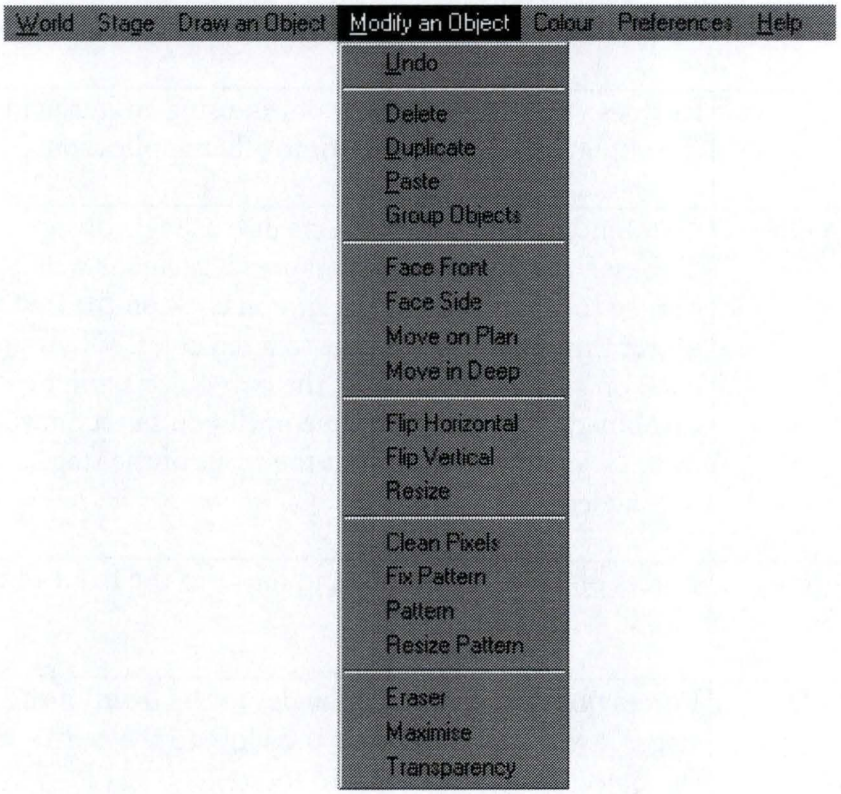


Figure 5: *Modify an Object* Menu

Table 4: Functionalities to modify an Stage Object (Menu *Modify an Object*)

Tool + = double-click on the tool	*Item*	*Description*	*Mac* (v.1.5)	*PC (v.0.5)*
	Undo	Undoes the previous command.	X	X
	Delete	Click on an object to permanently remove it.	X	X

Tool + = double-click on the tool	*Item*	*Description*	*Mac (v.1.5)*	*PC (v.0.5)*
	Duplicate	Create a duplicate copy of any object.	X	X
	Paste	Enables you to create a new object using an image in the clipboard copied from some other application.	X	
	Group Objects	Combines two or more objects into a single object. This operation cannot be undone. The cursor will change to a white cross. When you click on the first object the cursor will change to a red cross. When you click on subsequent objects, the two objects will be combined. This will continue until you select another tool. The objects must be at the front of the stage and not angled.	X	C
 +	Face Front	Forces object to face front and move to the front of the stage.	X	
 +	Face Side	Forces object to be at right angles to the front of the stage. It will avoid locations too close to the centre, as the object tends to be hard to locate.	X	
	Move on Plan	Enables you to move an object on a vertical plane from the front view. By selecting an object to be moved in this way, a dotted rectangle will appear indicating the vertical plane in which the object can move. As the distance from the front of the stage is fixed, the object will not get larger or smaller. If in Top View, the objects (represented by thin rectangles) can be moved around the stage by picking them up from the black square at their centre. Object can also be angled by picking up either end and dragging up or down the screen. The object will then 'rotate' around its centre. When you return to Front View, the object will appear in perspective.	X	X

	Move in Depth	Enables you to move an object on a horizontal plane from the front view. By selecting an object to be moved in this way, a dotted trapezoid will appear indicating the horizontal plane in which the object's centre can move. As the cursor moves away from the centre of the screen and towards the top or bottom, so the object will move towards the front or back of the 'stage' (and hence get larger or smaller).	X	X
	Flip horizontal	Clicking once on an object will cause it's image to be flipped horizontally (i.e. right becomes left and left becomes right)	X	X
	Flip vertical	Clicking once on an object will cause it's image to be flipped vertically (i.e. top becomes bottom and bottom becomes top).	X	X
	Resize	Enables you to resize an object. When clicking on an object a green rectangle will appear indicating its size (necessary for images shown as 'outline' or 'transparent'). When the cursor crosses this line with the mouse button depressed, the line will turn red and the object will be resized to the position of the cursor. If the shift key is held down when commencing this operation the original proportions of the object will be retained. When the button on the mouse is released the object's new size will be retained.	X	X
	Clean Pixels	Clean an image by (trying to) eliminate stray white pixels, especially around the edge.		C
 +	Fix Pattern	Makes a pattern become fixed. This enables the current pattern to be made into a component of a further pattern. The operation is forced if you use an eraser on a patterned object.	X	C
	Pattern	Allows the image on an object to be changed to a repeating pattern (or vice versa). Users click on the object and the image will be reduced so that it is a maximum of 60 pixels in any direction and is repeated to fill the same space as before. If appropriate, the patterned surface will be shown in perspective.	X	C

	Resize Pattern	If you click on an object with a pattern, then a resizing process will follow (see 'Resize' above). A red rectangle will indicate the size of the repeating pattern which can be resized.		C
	Eraser	Selects the eraser tool which enables you to erase any part of an image. The thickness of the eraser can be set by changing the pen size. (One way to do this is to double-click on the eraser tool.) The eraser will force the object to be transparent. The eraser tool can only be used on objects at the front of the stage and not angled (if necessary an option is presented to make them so).	X	C
 +	Maximise	Forces object to its maximum size (i.e. full stage height and full width or depth). The operation can only be undertaken on objects facing front or side.	X	C
	Transparency	If you click on an object, the transparency of the image will change, i.e. if it is currently 'transparent' (all white pixels are invisible) it will become 'opaque' (i.e. all white pixels within a white rectangle will be shown). If it is not transparent, it will become transparent.	X	C

- *Colour* Menu

Fifth, we have the functionalities that manage the colour characteristics of the software. The first piece of the menu is to express the current colour. The second enable the painting of stage objects or even the current stage/scene. The third allows the possibilities about the luminosity of the stage objects or even the current the stage/scene. The last are not implemented in the PC version. It allows specifying the place of a spotlight.

(See Figure 6 and Table 5)

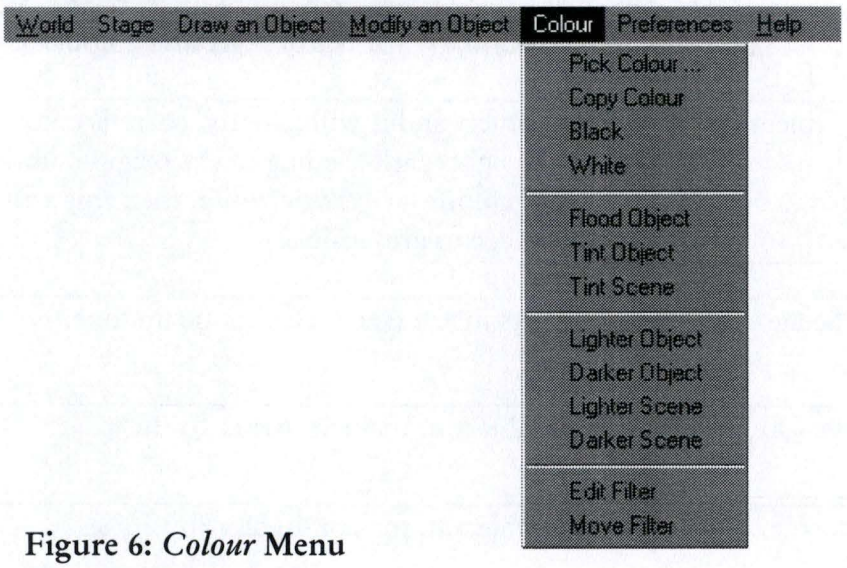


Figure 6: *Colour* Menu

Table 5: Functionalities about Colours (Menu *Colour*)

Tool + = double-click on the tool	*Item*	*Description*	*Mac* (v.1.5)	*PC (v.0.5)*
	Pick Colour ...	Opens the colour picker to select a new colour.	X	X
	Copy Colour	Make the current colour equal to the colour of the chosen pixel.	X	X
	Black	Set current colour to black.	X	X

Tool + = double-click on the tool	*Item*	*Description*	*Mac (v.1.5)*	*PC (v.0.5)*
	White	Sets the current colour to off-white (i.e. not transparent).	X	X
	Flood Object	Paste the current colour to an object. All non-white pixels in the object's image will be set to the current colour. If the floor is selected, it will be set to the current colour. Similarly for the background colour.	X	X
	Tint Object	Click on an object and it will take the current colour but retain the light/dark shading of the original image. If the current colour is black or white, then this will change the image to greyscale.	X	C
 +	Tint Scene	Tint all objects in the scene. Cannot be undone.	X	
	Darker Object	Click on an object to make it darker by 10%.	X	X
	Lighter Object	Click on an object to make it lighter (by 10%)	X	X
 +	Darker Scene	Darken all objects in the scene by 10%. Cannot be undone.	X	
 +	Lighter Scene	Make all objects in the scene lighter. Cannot be undone.	X	
 +	Edit Filter	Invokes a dialogue window that allows you to switch the filter on/off, set the colour of the light, set its brightness and angle of diffusion, as well as its shape (circular or elliptical)	X	
	Move Filter	Click on the screen and the centre of the light filter will move to where the mouse is clicked.	X	

We can suggest that the items of this later piece of menu are not in a nice position. A best position is the *Preferences* menu for the *Edit Filter*, and the *Stage* menu for the *Move Filter*.

- *Preferences* Menu

Sixth, we have the user's preferences. We can change the size of the pencil, and also some different customisable options: *Show/Hire Help text*, *Show/Hire Menu Bar*, *Show/Hire Position of Pointer*, *Show/Hire Time Line*, *Use/Not Use Transparency* and *Show/Hire Wireframe*. (See Figure 7 and Table 6)

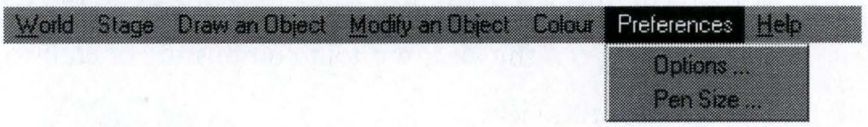


Figure 7: *Preferences* Menu

Table 6: Visual Assistant's Preferences (Menu *Preferences*)

Tool + = double-click on the tool	*Item*	*Description*	*Mac (v.1.5)*	*PC (v.0.5)*
 +	Options ...	Produces a dialogue box that will enable you to switch the menu bar, the tool panel, the *wireframe* and the help text on/off. It can also enable / disable the default option to move an object if dragging occurs when no appropriate tool is selected.	X	X
 +	Pen Size ...	Open a dialogue window to reset the size of the pen (for Pencil, Line and Eraser tools)	X	X

- *Help* Menu

This functionality is not implemented in the current version, but we have enabled the possibility to provide help in next version of Visual Assistant.

11) Conclusion

The use of computers only to retrieve information, or to detail drafting or modelling, may miss a great opportunity to explore in a deeper way within theatre and performance education.

On the one hand, Visual Assistant software can be seen as a more general visualisation tool for theatre. It is a 3D sketchpad or scrapbook that allows the possibilities to work with pictures to help us visually understand a play and to communicate simple ideas about a performance in a visual form.

On the other hand, this is a preliminary in a longer process to design a real scene. It is a way of working out the ideas without committing oneself to the time involved in building significant models.

We think that Visual Assistant software has something new to help professionals as well as educators and students in the field of theatre to improve their creative activities. We are convinced that the Visual Assistant software is a well-designed sketching tool for theatre practices and that it increases the freedom to communicate visually, but also to play visually, to act visually and to improvise visually.

Let us see Dominique Serron's remark about Visual Assistant software:

> *" The 'collage' effect of Visual Assistant gives a surrealistic impression to the scenes, whereas the real essence of theatre is impressionistic (i.e. transmitting the emotions of the work) "*[29]

Her observation is very relevant. Indeed, Visual Assistant has to reconstitute as well as possible the theatre universe, avoiding the modification of its nature. Now, Visual Assistant gives the feeling to make *surrealistic* art, whereas the real essence of theatre is to be an *impressionistic* art. It is true that surrealism does exist in the theatre field, still it does not make it its main base.

---

[29] « *L'effet de 'collage' donne à Visual Assistant une impression de surréalisme des scènes, alors que l'essence même du théâtre est l'impressionnisme (c'est-à-dire transmettre les sentiments de l'œuvre)* » coming from a conversation we had with Dominique Serron.

D Reusing of the existing knowledge

This chapter covers the different approaches for building new softwares by using other previous knowledge coming from old developments and libraries of reusable components. This is not a complete work about the subject but just an introduction in the field. In this work, we are dealing with a migration of a software running on a Mac platform to a PC platform. Besides this, while the Mac version of the software is designed by using the C language, the PC version will use C++ language. Our goal was, among others, to build a reusable software which is, in our sense, a suitable quality of every software done nowadays. We are conscious that a gap exists between the requirements of the user and the implemented system, which means that a kind of semantic degradation is inevitable. To capture the maximum of the underlying semantics and to catch some of the lost ones, one needs to explore the different methodologies and knowledge from other fields such as the reverse engineering. We propose to investigate the different fields, to have an idea about the contribution of each one and to be aware of the difficulties related to this work.

1) Software Reuse

Software Reuse "*... is the use of programming language source text or compiled code in a context other than for which it was originally written* "[1], but where each reusable part has always the same original goal.

The software reuse exists since the beginning of the computer science. We find different means for doing that: publication of algorithms and ideas, high-level programming languages, and packages.

The publication of algorithms and ideas has been very important for the development of computing. The presence of standard textbooks in a sector is an indicator of its maturity, and is an important tool through which the reuse of ideas takes place.

The most obvious example of reuse is the use of high-level programming languages. In such languages, many frequently used combinations have been packaged into single constructs at the higher level[2]. Thus the high-level language provides a special notation for selecting generic constructs, instantiating them with the appropriate parameters, and finally composing them to build software systems.

The packages have also been an important aspect of the software reuse. As the publication at the first point, the procurement of standard packages is an important example of reuse, and the availability of a wide range of products is again an indicator of the maturity of a sector of technology. The packages that are defined rigidly are seldom useful. Some flexibility is essential providing through a range of capabilities, from the simple parameterisation, the configuration following some elaborate build script[3],

---

[1] Bott F. and Ratcliffe M., Reuse and design, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., Chapman & Hall, London 1992, pp. 35-36

[2] For example, the subroutine entry and exit with parameter passing.

[3] This is usually done for making an operating system.

until the modification of the package at the source level. A package may be seen as a form of software component[4].

In our case, we have used a high-level programming language using the *object orientation*. The object-oriented languages embrace the concepts of *encapsulation*[5], *messaging*, *classe*[6], *polymorphism*[7] and *inheritance*[8], and make code reuse easier.[9] These different concepts of this language are very useful to create interesting implementation architectures with reusable components, but also for capturing the representation of a domain. The oriented-object models a domain by identifying entities and their behaviour.

With the oriented-object the structure of the system is based on interactions between entities. The functions and the data are grouped together into classes according to abstractions that they represent. The objects are like people who communicate, enacting their behaviour of their job, providing different services to each other. And the architecture is the mean for allowing the co-operation between the objects.

Designing reusable components is not just a matter of deciding what data and functions should be encapsulated. Components only become usable in the context of architectures. Architecture does not come out of nowhere, but evolves from agreements on computational mechanisms, paradigms and other inventions that they have underwent evolution.

---

[4] The term *component* is very wide: it includes executable code, but also more abstract objects such designs, algorithms, specifications and requirements.

[5] Information and/or implementation details which are hidden.

[6] Describes the form and the behaviour of objects, and it is used as a disc matrix, a canvas, a mould for making as many objects as we want that have the characteristics described in the class.

[7] A code is said to be polymorph if it can be transparently used on instances of different types. Polymorphism allows us to write code in terms of generic type and have it work correctly for any special type issued from the generic type.

[8] Concept of hierarchy allows programmers to reuse not only the implementation, but also the interface of another object.

[9] See chapter *Software Architecture.*

The architectural view focuses on *frameworks*[10] of generalised ideas and sees reuse as an issue of *plugability*. Compatibility with the architecture of the new application requires our components to behave consistently with other components. A component becomes usable when it fits into an architecture, and to be reusable it must possess the following qualities: it anticipates other contexts of use, the component is *findable*, it is able to be understood, all the services it requires can be provided.

[10] A set of co-operating classes that makes up a reusable design for a specific class of software. A framework provides architectural guidance by partitioning the design into abstracts and defining their responsibilities and collaborations. A developer customises the framework to a particular application by *subclassing* instances of framework classes.

2) Software Migration

Software Migration is the migration of software from "... *an existing source system to an intended target system*"[11] by using "... *a mixture of activities such as the application code conversion, reengineering of user interfaces, and portability of application components.*"[12]

Software Migration takes place between an existing *source* system to an intended *target* system. A system can be an *operating system*, a *machine* (Power PC, Intel, AS400, ...), or even a *virtual machine* such the virtual machine Java.

It is a tedious, expensive and time-consuming effort that must be managed carefully. It may take anywhere between six months to two years.[13] If done correctly, migration can produce significant payoffs.

System migration is a good opportunity to rearchitect. Rearchitecture, if chosen, requires starting from scratch with requirement gathering and project planning. Rearchitecture involves a combination of application software architectures, and user interface reengineering. We can make code conversion in the way to adapt and/or to improve the software to migrate with the new technologies such object-oriented architecture.

Object-oriented technologies can play a key role in migrations by using object wrappers around the components being transitioned. They allow portability of application components.

---

[11] Umar Amjad, *Application (Re)Engineering : Building Web-Based Applications and dealing with Legacies*, Prentice Hall, Piscataway (New Jersey) 1997, p. 535

[12] Ibid., p. 534

[13] Ibid., p. 535

3) Reverse Engineering

Reverse Engineering is "*... the process of going backward through the development cycle*"[14], in the aim to "*... extract[ing] the specification*[15] *of a system from its functional objects and to store it and all relationships...* "[16] like to "*... generate new information about software such as synthesising abstractions...* "[17] "*The extraction of documentation*[18] *and higher-level description of software* [comes] *from the code itself.*"[19]

We can break the reverse engineering in a family of tasks with three levels: at the implementation level, at the design level, and at the business level. In this work, we are interested by the implementation and the design levels.

The implementation level is concerned with documenting code characteristics such as program structure, control flow complexity, internal data complexity and standards violations.

The design level is attended with documenting design characteristics such as modularity, coupling, cohesion, data and file design complexity.

---

[14] *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., Chapman & Hall, London 1992.

[15] i.e. "*...the embedded knowledge...*". Holloway S., Re-engineering Business Systems to use the next generation of software, *Software Reuse and Reverse Engineering in Practice*, p. 275

[16] Ibid., p. 275

[17] Müller Hausi A., Reverse Engineering Strategies for Software Migration, *ACM*, (?Canada?) 1997, p. 660

[18] i.e. the *redocumentation* principle that "*... is the production of semantically equivalent representation (on paper or not) of the target system at whatever level of abstraction is being addressed.* " Frazer A., Reverse engineering - hype, hope or here?, *Software Reuse and Reverse Engineering in Practice*, p. 215

[19] Hall P.A.V., Editorial Introduction, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., Chapman & Hall, London 1992, p. xiii

The business level is attended with documenting in a non-procedural way the business functions that a system performs. Therefore the descriptions obtained are design independent.

A key aspect of reverse engineering is that with a code only system, the process is bottom-up. Reverse engineering strategies have become used in environments, which require levels of documentation top-down. Such strategies raise the question of information validation. For example, how can you validate the higher-level *data flow diagrams*[20] if the lower levels of information are not available?

a) Difficulties of reconstructing architectures

When we develop a system, the evolution of a lot of software properties is really critical to have a success with the product. However, reasoning about the *intended* architecture of a system is distinct from reasoning about its *realised* architecture. When we design and eventually proceed to the implementation of architecture, faithfulness to the principles of the intended architecture is not always easy to achieve. This is particularly true in cases where the intended architecture is not completely specified, and documented or disseminated to different members of the whole project.

This problem is exacerbated during maintenance and evolutionary development, as a drift and erosion of the architecture. However, if we wish to transfer our reasoning about the properties of the intended architecture of a system to the properties of the implemented system, we must understand to what degree the realised architecture *conforms to* or *is like* the intended architecture.

We can measure the architectural conformity if the two architectures to be compared are available: the intended architecture and the architecture that is realised in the implemented system. The former[21] should be documented early in the system life-

---

[20] " *A dataflow diagram is a partial image of the way of working of an information system or from a part of the information system; it shows graphically the production, the flow and the destination of messages in the organisation* ". Bodart F. and Pigneur Y., *Conception assistée des systèmes d'information : Méthode – Modèles – Outils*, Masson, Paris 1989, second edition, p. 98

[21] That is the *ideas man* (see appendix *French Translator*) i.e. the person who begins the development of the system.

time and maintained throughout all the process. However, the latter typically exists only in the artefacts such as source code, makefiles[22] and, occasionally, designs that are directly realised in the code[23]. In addition, it is unusual that an implementation language provides explicit mechanisms for representation of architectural constructs. Therefore, facilities for the reconstruction of a software architecture from the artefacts is critical in measuring architectural conformity.

Beyond this point of measuring architectural conformance, software architecture reconstruction also provides important leverage for the reuse of software assets. The ability to identify the architecture of an existing system that meets some goals fosters reuse of the architecture in systems with similar goals. In other words, if we find what the architecture of an existing system does, we can take it for developing a new architecture. Consequently, *architectural reuse* is the *cornerstone* practice of product line development that keeps the acquired knowledge of the past.

b) Static information is insufficient

A significant quantity of information may be extracted from static artefacts of software system such as source code, makefiles, and design models, using techniques that include parsing and lexical analysis. Unfortunately, system models extracted using these techniques provide a minimum of information to describe the run-time nature of the system. The primary factor contributing to this deficiency is the widespread use of programming language features, operating system primitives and middleware[24] functionality. From that the specification of many aspects of system has been deferred until the run-time analysis. These mechanisms permit systems to be designed with low coupling and a high degree of flexibility. But they obscure the architecture reconstruction process.

---

[22] Used to define the compilation process for a software project.

[23] By using, for example, an architecture description language.

[24] • A term used to describe software that sits below the application and hides the different operating systems, databases, network systems and protocols. It provides implementation independent programming interfaces for an application.

• Medium layer software, i.e. more abstract and at a higher level than traditional operating systems, providing an infrastructure for applications.

In particular, static extraction techniques can provide only limited insight into the run-time nature of systems constructed using such techniques, because many of the details determine actual communication and control relationships simply do not exist until run-time, and hence cannot be recognised until run-time.[25] However, for doing this work, one needs a specific tool for the reconstruction and it does not exist on the market because it is not easy for such tools to gather such information. It is very useful to exploit the abstract syntax trees, using parsing techniques such as in compiler approaches. Furthermore, there are a lot of products on the market that use these techniques.

Now, we must see how we may extract dynamic information from running system. Some techniques to accomplish this, include profiling and user defined instrumentation. The first technique is used for system performance analysis. To use this method, one can compile a specific system with a special flag that instructs the compiler to instrument the code such that it records information coming from function invocation during execution. After that, the system is exercised and recorded information is analysed. This technique may be used to determine actual function invocations, augmenting our statically extracted models with improved information concerning polymorphic functions and functions executed through function pointers.[26]

In a similar fashion, the second technique, which is user defined instrumentation, is a mean for adding special purpose tracing functionality of a system to allow monitoring its operation. For example, instrumentation can be added to the application code responsible for interprocess communications to determine the system run-time communication. In addition, it is sometimes possible to instrument libraries, or even the operating system. This allows the instrumentation of systems without modifying any application code and requires less application specific knowledge.

---

[25] For example, relationships among communicating processes might be determined via an initialisation file or even dynamically based upon the availability of processing resources.

[26] For example, in C or C++.

c) Reconstructing architecture in a vacuum

Unfortunately, it is frequent by the case that the efforts to reconstruct the software architectures of systems must contend with a complete lack of pre-existing architectural information. This often occurs when the system being analysed is particularly old. There are no longer any designers or developers that can relate architectural information, or the system intended architecture was never documented. Furthermore, these are typically the situations in which we are most interested in recovering an architecture. In particular, it is common for such systems to be involved in ongoing maintenance or even undergoing a more global re-engineering effort as modernisation or porting, for example.

Successful architecture reconstruction revolves around the acquisition of architectural constraints and patterns that capture the fundamental elements of the architecture. Regardless of the mode of development of a system, its evolutionary state, or its age, such constraints and patterns are always present. However, they are rarely captured explicitly even when a truly architecture based development process is followed. Thus, the primary task of the reconstruction analyst is the acquisition of this information by means other than search for documentation.

4) Forward Engineering

" Forward Engineering is the process by which some abstract high level specifications obtained by the reverse engineering are used to reconstitute a system in a new form. "[27]

We can divide the forward engineering in a family of tasks in three levels: *code re-implementation*, *application re-design*, and *revision of the specifications* with development of a new application architecture.

In all cases, the forward engineering is the last thing when dealing with re-engineering. At the beginning, we use the reverse engineering to have a degree of abstraction of a system. At the end, we use the forward engineering for reconstituting the studied system to another improved form.

Furthermore, we choose the forward engineering methods with regards to the reverse engineering methods used. In other words, the level that we use in the forward engineering depends upon the level that we have used in the reverse engineering to study the system.

A lot of problems can be solved thanks to the forward engineering phase, by firstly classifying the specifications found during the reverse engineering work. For example, the complex procedure control flows, the complex internal data usage, the poor program structure, the poor design decomposition, the poor data or file design across one or more programs and databases, the hard coded boundaries, the standards violations... can be all eliminated. A successful forward engineering must be able to remove clearly defined types of problems.

---

[27] Ob. Cit., *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V.

5) Re-Engineering

" Software Re-engineering is the process of examining and altering a subject system to reconstitute it in a new form. "[28] i.e. it *"... is the process of reverse engineering a subject system to a chosen level of abstraction and then reconstituting the system by means of forward engineering. "*[29]

Re-engineering can be broken into two phases: the first is the reverse engineering and the second is the forward engineering. Of course, we can break these two phases into a family of activities.

Re-engineering concerns some forms of *restructuring* work at the program level, and the reverse engineering is the process of abstracting back to some high level specification. But, these notions lack of strictness, and it is not easy to find a simple classification of all the terms. In fact, all these notions are not independent and depend on each other. For example, even to re-engineer a single program it is necessary to reverse engineer it to a higher level of abstraction than the code level. Then it is necessary to forward engineer it in a new application.

---

[28] Art. Cit., Müller Hausi A., Reverse Engineering Strategies for Software Migration, p. 659

[29] Art. Cit., Frazer A., Reverse engineering - hype, hope or here?, *Software Reuse and Reverse Engineering in Practice*, p. 215

6) Code translation

" Code translation is mechanical extraction of a description of an application and implementation of it in another language. "[30]

Code translation is a transcription *line by line*. Of course, it enables some little obligatory own adaptation of the target language, but we do not rethink the source code. The new system will reflect the code complexity, design errors and architecture errors contained in the original. The re-engineering and code conversion are not a code translation:

- The re-engineering implies some significant changes to give an improvement of the original.
- The code conversion implies redesign, adaptation and/or improvement of the original code.

[30] Ob. Cit., *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V.

7) Conclusion

We have surveyed software reuse, software migration, reverse engineering, forward engineering, re-engineering and code translation. Reuse is implemented with accumulating libraries of components, in which we can have designs, specifications, as well as code. Migration is a set of different activities, which have the same aim to take a software from an existing source system to an intended target system. Reverse engineering is a method for obtaining components in order to assist in the understanding of code before making changes. Code translation is mechanical extraction of a description of an application and its implementation in another language, like an adapted *cut and paste*.

To improve a complete system, or to transfer it into a new operational environment, we can firstly use reverse engineering, in order to have all the knowledge about the system, secondly, we can use re-engineering to complete the task.

In reuse, the major problem is the component description, component matching, and component adaptation. In software migration the major problem are an expensive undertaking that may take anywhere between six months to two years and the need to consider for expected applications to support critical business functionality for long time periods. In reverse engineering the major problems are the use of the informal information of the natural language embedded in comments and documentation, and also the systematic use of testing.

Ten years ago 4GL[31]'s were seen as a technical solution to a more general productivity problem. Apart from a mixed result from using 4GL's, most 4GL users now find that they will have great difficulty to re-engineer 4GL systems because there are no reverse engineering tools available to support these languages. Some 4GL vendors claim that the maintenance work is cut by up to 90%. But the inability to reverse engineer a 4GL system means that the user may face the ultimate maintenance crisis which a 4GL was presumably meant to avoid. The dilemma is to throw away a system and most of the knowledge it contains, or to redevelop a replacement from scratch.

First, Visual Assistant was developed by Colin Beardon for Mac platform with C language. Second the previous year, Frédéric Miche and Colin Beardon have worked on the first PC version, which is written in C++ Builder. Third, we have worked on the second PC version by using the two previous works.

In this work, we have used a mixture of all the categories of reuse knowledge, that we have described in this chapter. In fact, we have used the *software reuse* in order to make use of the existent functionalities and the available algorithms. It is obvious to use the *software migration* because it is the essence of our work, which is to migrate a Mac software to a PC software.

We have also made *re-engineering*. First, we have made *reverse engineering* because the two first versions did not have any documentation or specification. Second, we have used *forward engineering* to reconstitute the software in a new form by using the synthesising abstractions coming from the *reverse engineering*.

Finally, we have made some *code translation*. C++, which is based on the C language, is the programming language used for the PC versions. Note that C is the programming language for the Mac version.

---

[31] Fourth Generation Languages i.e. languages that use *automatic painters softwares* to draw the different application windows and also their graphical/visual objects. The programming tool used for migrating Visual Assistant to PC platform is *C++ Builder*: it is a 4GL.

E Software Architecture

This chapter will present the organisation of the software developed in this work. In other words, we will explain the interaction of the different components of the system. The software was originally designed for Macintosh platform; it had to be migrated to a PC platform, the reasons behind this migration seem to be the relative importance of the PC market compared to the Macintosh market. Indeed, the PC market represents 90% of the microcomputing world market.

We will also investigate the method adopted in this work, to represent on computer monitors, which are two dimension systems, objects originated from the reality, which is three dimensions. We have already established in the chapter dealing with virtual reality, that it is possible to design interfaces allowing a three-dimension sight, it is the case for example, when electronic helmet with visor is

used. This kind of device is very expensive and not really spread, it is why we have opted for a traditional representation on our traditional two-dimension monitors. Besides that, the monitor is very well known by the public and can be found everywhere. All these reasons make monitors familiar to people more than any other device.

The software designed here was originally developed in C language, and while doing its migration, it was asked to convert it to C++ language. The reasons of this choice will be explained as well as some fundamental concepts of C++ according to the Satir and Brown's book[1].

[1] Gregory Satir and Doug Brown, *C++: The Core Language*, O'Reilly, USA 1996.

1) The object-oriented programming

The language chosen for the development of this Visual Assistant software is C++. This language extends the popular programming language C to support object-oriented programming[2]. To understand C++, we must first understand object-oriented programming and what C++ adds to C to support this new way of programming. But this is not the aim of this work. We suppose that the reader knows this new way of thinking in the field of computer science.

Object-oriented programming is not very new. We do not need an object-oriented programming language to program using object; this can be done in C language by simulating some concepts if necessary, as did the first C++ implementation.

But a language like C++ makes object-oriented programming easier, because it directly supports the creation and use of objects. Moreover, it obliges the programmer to have an obligatory canvas allowing a better programming. In fact, in C++, it is obligatory to use and to think in object-oriented.

For the understanding of the design of the architecture of the software, We will describe, in a brief way, the fundamental object-oriented functionalities. We consider these functionalities as the most important and interesting ones of C++ for a good development of the application designed in this work.

a) Important Object-oriented concepts use by C++

Let us describe the important concepts of the C++ language, and try to compare it with the C language. In fact, as in this work we deal with the migration of a Mac software written in C to a PC software designed in C++. It is worth starting with a comparison between the two languages to demonstrate the reasons underlying the change of language.

---

[2] Object-oriented programming focuses on the objects that make up a program, rather than the functions and the data.

To be more formal, object-oriented programming languages usually support a few key features. Every object philosophy includes a slightly different set. But, we can describe the more important ones:

- The first is the *abstraction.*

This notion means the creation of a well-defined interface[3] for an object. Proper abstraction separates the implementation of an object from its interface. In C language, abstraction involves wrapping a data structure with a functional interface. And a whole object is created by data and functions together.

The C++ language recognises this relationship by allowing us to group data and functions that use it together using a *class*. The class is the fundamental addition of the C++ language, with regards to the C language.

On one hand, the class allows the standard concept of abstraction thanks to the principle of protecting its private information from access by other classes.

On the other hand, several classes can collaborate so as to make a complex abstraction by sharing private information without allowing other classes to violate the combined abstraction.

A class describes the form and the behaviour of objects. It is like a stamp, or even a matrix, out of which we can press as many objects as we want. A class is essentially a new type we are adding to the programming language.

- The second is the *encapsulation.*

This means keeping the implementation details of our abstractions private, i.e. hidden outside of the scope of the implementation just to avoid a bad access to these details. Proper encapsulation both encourages and enforces the hiding of implementation details.

Moreover, it makes our code reliable and easier to maintain because we know exactly what can be done with the abstractions that we implement. In fact, all

[3] An interface defines a list of methods, that the user can see and use, by hiding the complexity of the implementation.

is very centralised in our abstraction, which is like a bubble that separates the internal parts of our abstraction from all other parts of the program.

The class is the unit of encapsulation. The class is encapsulated through *access control*, that is, by controlling who has access to the class members. As programmer, we can declare which class members are accessible to users of the class and which are accessible only to the class itself. The members accessible outside the class create the interface to the class, this is the capsule that encloses the implementation details.

Each *access control specifier* determines which users can access the class members as follows. First, the class members following a *public* label can be accessed from any user by means of the functions[4] that are in this public part of the class. These create the *class interface*. Second, the class members following a *private* label can only be accessed inside the member functions of the class[5]. Private members create the *class implementation*. Third, there is also the access control specifier *protected*. The class members following a protected label are like the class members following a private label except they are accessible by the *derived* classes[6]. If we want a good object-oriented programming, we cannot use this last specifier. In fact, this one kills the protection allowed by the C++ language, and we can program without it by using just the two *specifiers* public and private.

We can simulate that in C language by separating the interface functions and the implementation of them in two different files. In this manner, the users of the functions use the file with the interface and cannot see the hidden implementation.

---

[4] It is possible to place also public data, but it is not really nice in the object-oriented programming because that kills the encapsulation principle.

[5] Also by the *friends* functions. A class can make an individual function a friend. This function then has access to the class private members without itself being a member of the class. This principle allows the possibility of two or more classes to work together very closely by co-operating to create a single abstraction. Friendship can be misused as any features that let us suspend a language's normal security. We can use friendship to create a unified interface out of more than one class, never as a work-around for improperly designed abstractions.

[6] Derived class are defined in the next point about the hierarchy.

When we see this tip, we can say that the C++ language has drastically improved this primitive form of encapsulation.

- The third is the ability of having a *hierarchy*.

When we make a new abstraction, we often have in mind that we have made previously the same abstraction, but maybe with some different details. It could be really nice if we could be able to reuse some good abstractions without rewriting them.

We can reuse a good abstraction as a basis of many other abstractions. In this way, we create hierarchies of abstractions. Object-oriented programming provide three mechanisms for building these hierarchies: *composition*[7], *derivation*[8] and *templates*.

The *composition* allows us to create objects with other objects as members. We call the principle composition because we compose larger objects out of smaller ones, and we have the impression of inserting the small objects to make bigger ones.

The *derivation* allows a class to use not only another class for its implementation, but also to share its interface. Of course, it is possible to simulate derivation in C language, but is not common.

Does derivation reduce the protection by encapsulation? The power of derivation is that it is a close form of sharing that maintains encapsulation between the two classes. It means that if we derive a class from another one, called *base class*, it is impossible for the new created class, called *derived class*, to access any information that is member of the private part of the base class.[9]

In the last way, the *template* allows programmers to make a definition of a class with parameterised parts. We can write code in C language that does the same by using *casts*[10]. We can use cast in C++ too, of course, but that is not type-safe.

---

[7] Alternatively *aggregation* or *layering*.

[8] Or also *inheritance*

[9] But it is possible if we use the protected access control specifier in the base class.

[10] To convert a variable from one type to another type by explicitly indicating the type convertion.

Using templates to manipulate an array of any type allows flexibility without sacrificing type safety. So templates allow a more powerful kind of hierarchy.

- The last is the *polymorphism*.

Polymorphism allows us to use objects of different classes of a same hierarchy in the same code. In other words, this polymorphic code can be transparently used on instances of different classes. A typical example of this notion is a group of different planes: rectangles, ovals, squares, circles, triangles, etc. Each shape knows how to draw itself, calculate its area, and so on. Of course, every type of shape does this differently, but they all share this ability. Polymorphism allows us to write code in terms of generic shape type and have it work correctly for any actual shape.

The generic shape is polymorphic[11] because at any particular point during program execution it can be a rectangle, an oval, etc. Polymorphism requires the generic shape to behave differently at run-time depending on the actual type of shape.

The dynamic typing nature of this mechanism is difficult to mesh with the static typing nature of C language. Of course, we can realise the polymorphism in C language, but with some degree of *hackery* because the C language does not directly support polymorphism.

To achieve the switch at the run-time between the good implemented functions with regards to the run-time type of the object pointed, we use polymorphic functions[12]. We need this sort of functions because the compiler cannot know the run-time at the compilation. If we do not make our function as polymorphic, the executed function will always be the function of the generic class and not the function of the class of the real object[13]. Why not make, by default, all the functions polymorph as in Java? Because the virtual functions are a little slower than the traditional functions.

---

[11] Literally *many forms.*

[12] Also called *Virtual Functions.*

[13] The compiler will solve the executed function at the compilation.

b) A method to implement an object-oriented design

At this point we discuss how to implement an object-oriented design. Object-oriented design involves partitioning classes, and assigning responsibilities to them. We do not discuss object-oriented design details, but generic object-oriented design of any object-oriented languages.

The design is a critical step in developing code, especially object-oriented code. We cannot save time by cutting corners during object-oriented design if we do not want to loose time during implementation, debugging and maintenance. In fact, object-oriented design is the foundation of our application, as the foundation of a building. When we construct the floors of a building, it is very complicate to change the base of the construction if we have a problem unless by destroying the building.

When we have defined classes and their functional interface, there are a variety of ways to implement these classes. That means that it is not just a matter of coding, and that the differences of coding can be subtle, and indeed critical. We must remind the different possibilities of object-oriented language: templates, derivation, composition, and polymorphic functions. But, when should we use this principle or this other principle? We discuss here a technique for deciding how to make this decision among others. This approach is not just the only way to do; there are a lot of other manners. More decisions can be swayed by far too many special circumstances that we cannot summarise here.

To see a method is really nice to have an idea of the subject. We see in Table 1 on the next page a summary of the method. Moreover, we find an example for each case, in order to understand the theoretical approach.

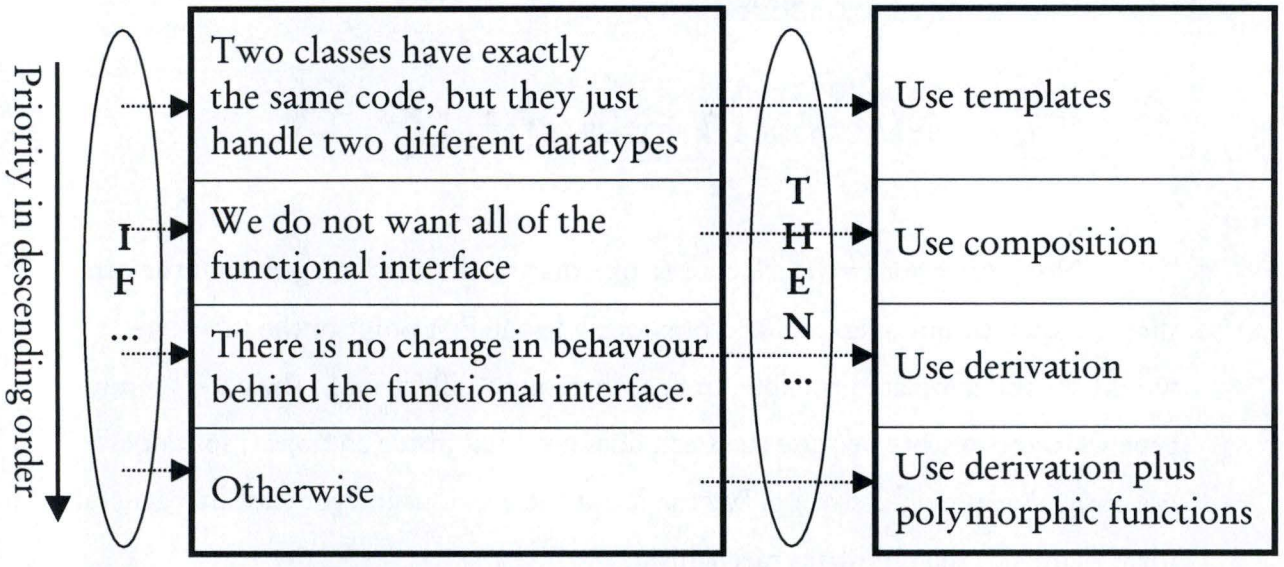


Table 1: Choosing the Relationship between Two Classes

- *Two classes have exactly the same code ⇒ Use templates*

The first question is whether the two classes can have the same code applied to different types. If so, we can use the template mechanism. An example is a stack of integers and a stack of pullovers. The code for `push()`, `pop()`, and any other member functions of the stacks will be the same except for the type of the object they are manipulating.

Example: we can illustrate this table with a stack class. Its functional interface offers four functions that we can apply to the class: `Push()`, `Pop()`, `Clear()`[14], and `PrintOnScreen()`. For simplifying the explanation, we define a simple class with no dynamic. In fact we use a static table to keep the data of the stack and we also use a static number of elements in the stack defined by STACK_SIZE.

```
class IntegerStack
 { public:
     IntegerStack(void) { NumberOfItems = 0; }
     void Push(int Integer);
     int Pop();
```

[14] Delete all objects of the stack.

```
    void Clear();
    void PrintOnScreen();
  private:
    int NumberOfItems;
    int DataOfTheStack[STACK_SIZE];
};
```

Now, we want two stack classes, one that works with integers, and the other that works with pullovers. As said previously in the first point of the table, we would use the template principle. In fact, there is no difference in the code for any type we want to store because the stack does not look inside an object, so it does not care what type it is storing. We can adapt the previous integer class in a general form thanks to the templates mechanism.

```
template<class kind>
class Stack
 { public:
    Stack(void) { NumberOfItems = 0; }
    void Push(kind Integer);
    kind Pop();
    void Clear();
    void PrintOnScreen();
  private:
    int NumberOfItems;
    kind DataOfTheStack[STACK_SIZE];
};
```

We can see above that the `Stack` class is a class of `kind` type. Of course, `kind` is not a type, but can catch all the 'realist type' to make a stack of the specified type. In our case, we can make the two different classes, which we would want, by specialising[15] the general form of the template.

```
Stack<int> OwnIntegersStack;
Stack<pullover> OwnPulloversStack;
```

[15] *Specialisation* by opposition to *generalisation.*

- *We do not want all of the functional interface ⇒ Use composition*

The next question is whether a class, called *using class*, uses the entire functional interface of another class, called *providing class*. Of course, the derivation principle might work if we want to add all functions from the functional interface of the providing class to the functional interface of the using class, but not if we want to subtract some functions from the functional interface of the providing class.

At the opposite, we use the composition concept, which is making an object of the providing class a data member of using class. In this manner, we can define whatever functional interface we want for the using class. In fact, it is not obligatory with the composition concept to put the functions from the providing class.

Example: we can suppose that we want an unprintable stack. It would have `Push()`, `Pop()`, and `Clear()`, but not `PrintOnScreen()`. The code is not the same for the two classes, so we can go to the second question. We do not want the entire functional interface, so we use the composition technique as specified in this second question:

```
class UnprintableIntegerStack
 { public:
     UnprintableIntegerStack(void) {};
     void Push(int Integer);
     int Pop();
     void Clear();
   private:
     IntegerStack Stack;
 };

void UnprintableIntegerStack::Push(int Integer)
    { Stack.Push(Integer);
    }
int UnprintableIntegerStack::Pop()
    { return Stack.Pop();
    }
UnprintableIntegerStack::Clear()
    { Stack.Clear();
    }
```

As we can see, we have removed the `PrintOnScreen()` function and defined the rest by calling the corresponding function for the instantiated `IntegerStack` object placed by composition in the new defined stack above.

Now, if there happens to be any polymorphic functions[16] for which we do not want the default behaviour, then we need to define an *intermediate class* between the using class and the providing class. This intermediate class derives from the providing class and allows to define the new behaviour of the polymorph functions which are properly defined in the providing class. Then we use composition mechanism by making an object of type of the intermediate class a data member of the using class.

Example: we can suppose that we want, as in the previous example, an unprintable stack, but we do not want to delete all the object of the stack when we call the `clear()` function. We want just to delete the first object in the stack, but in opposition to the `pop()` function, this new behaviour for the `clear()` function does not return the object it deletes.

The intermediate class:

```
class UnclearableIntegerStack : public IntegerStack
 { public:
     UnclearableIntegerStack(void) {}
     void Clear(); // overriding polymorphic function
   private:
     IntegerStack Stack;
 };

UnclearableIntegerStack::Clear()
    { Stack.Pop(); // We do not return the value that
    }              // Pop() takes out of the stack.
```

[16] Pure or not.

The providing class needs its `clear()` function to be polymorphic. In fact, it is only possible to override the polymorphic function, what we cannot do with the static and traditional functions. In this way, we prefix the function by the keyword `virtual` in the providing functional interface of the class.

The providing class:

```
class IntegerStack
 { public:
     virtual void Clear();
      …
   private:
         …
 };
```

The using class becomes:

```
class UnprintableIntegerStack
 { public:
     UnprintableIntegerStack(void)
         { NumberOfItems = 0;
         }
     void Push(int Integer);
     int Pop();
     void Clear();
   private:
     UnclearableIntegerStack Stack;
 };

void UnprintableIntegerStack::Push(int Integer)
    { Stack.Push(Integer);
    }
int UnprintableIntegerStack::Pop()
    { return Stack.Pop();
    }
UnprintableIntegerStack::Clear()
    { Stack.Clear(); // delete the first integer
    }                // in the stack
```

- *There is no change in behaviour behind the functional interface*
  ⇒ *Use Derivation*

The last question in the table asks whether there is any difference in the behaviour of any of the functions that a class, called *derived class*, derives from another class, called *base class*. If there are no difference in the behaviour then we have a normal derivation mechanism. We can add functions and data to the derived class, but there are no changes to the functional interface or even the behaviour of those functions defined in the base class.

Example: Suppose we want a reversible stack. The code is not the same as `IntergerStack`, so we cannot use the template mechanism. We want to keep all the functional interface of `IntegerStack`, so we do not have to resort to composition principle. We do not want to change any of the behaviour behind functional interface of `IntegerStack`, so we can use the normal derivation system. We just want to add another function, `ReverseStack()`:

```
class ReversibleIntegerStack : public IntegerStack
 { public:
     ReversibleIntegerStack (void) {}
     void ReverseStack(void);
 };
```

- *Otherwise ⇒ Use derivation plus polymorphic functions*

If the derived class needs a different behaviour for at least one function derived from the base class, then we need polymorphic functions. In this case, the polymorphic functions need to be there already in the base class. If not, we need to adapt the base class to make these functions polymorphic. If we are using a class from a predefined library, and the functions we want to change are not polymorphic, then we cannot use the derivation mechanism. We must use the composition.

Example: we want a Roman numeral stack. That is, it is still a stack of integers but printed as Roman numerals. Again we want to keep all of the interface functions, but we want to change the behaviour of one of them, `print()`. Of course, we need `print()` to be a polymorphic function in the functional interface of the `IntegerStack` class, or even we have to adopt the `IntegerStack` class to change the characteristics of `print()` function to make it polymorphic. If it is all right, we can derive the new `RomanIntegerStack` from the base class `IntergerStack`.

The derived class:

```
class RomanIntegerStack : public IntegerStack
 { public:
     RomanIntegerStack(void) {}
     void Print(void);// overriden only if print() is
 };                   // polymorphic in the base class
```

The base class:

```
class IntegerStack
 { public:
     virtual void Print();
      …
   private:
         …
 };
```

c) Why using C++ language and not C language?

We can get into a lot of trouble with C++ language if we are not careful. In fact, its concepts are very complex and powerful, and we need some experience of practising it. Why then would we use C++? We find that when pushing the computer to its limits, we can write code more elegantly in C++ than in C. It allows programmers to centralise various tasks, the most typical being memory management. C language can compete in run-time performance, but the code usually ends up looking pretty ugly. Other languages can compete in elegance, but usually with a significant run-time performance penalty[17]. As in the developed software *Visual Assistant*, fast and elegant code is really important, we have used C++ language.

d) And, what about the Java language?

There are a lot of differences between C++ and Java. But, only four differences between C++ and the traditional use of Java, are very important.

Two differences are in consideration of Java: in its traditional use, Java provides the possibility to have a portability of the run-time and, in all uses, it forbids the direct manipulation of the pointers helping to avoid bugs. The other two differences go in consideration of C++: the speed of the run-time is really fast, by comparing with the traditional use of Java, and it allows the direct manipulation of the pointers for greater flexibility.

What are the two kinds of use of Java? The first, which is the traditional one, allows the portability not only of the code, but also especially of the compiled[18] run-time. The latter allows an acceptable speed.

In the first case, Java makes a semi-compiled run-time. In this case, the code is compiled in a pseudo-code that is directly executed on a virtual machine. [19] By definition of a virtual machine, the semi-compiled Java code can run on all operating systems that have their specific Java virtual machine installed. This virtual ma-

---

[17] Like Java, Smalltalk, etc. We discuss about Java language in the next point.

[18] We murder the language with the *compiler* term. In fact, we use in the meaning: *compiled* and *linked.*

[19] Of course, it exists Java operating systems, but there are used by few people.

chine translates the semi-compiled instructions in the owner language of the operating system that runs below the virtual machine. This manner allows the portability of the applications without being slaved over the provider's operating system and without making different versions for every customer using a different system.

In the second, Java makes a compiled run-time of our application. This run-time depends upon the machine and cannot be transferred on another operating system. We need a specific compiler for each operating system.

In fact, in the case of our Visual Assistant application, we need speed, i.e. small answer time. Indeed, we use permanently graphical objects. These objects require a lot of computing time. We can advance that Java, which is semi-compiled is twenty times slower than the C++.[20] But, one can say, why do not use Java in its compiled version? If doing so, we just should loose the principal advantage of Java, i.e. its run-time portability. Besides that, even in its compiled version, Java is slower than C++. Let us notice that it is established that Java allows less programming errors than C++. This property of Java is derived from the fact that it is impossible to manipulate directly the content of pointers.

In summary, C++ permits us a big speed for Visual Assistant, and a great flexibility, mainly through the direct manipulation of the content of pointers. And at the opposite, Java allows us to have a portable run-time on different platforms.

From my personal point of view, it is be more suitable to develop the PC version of Visual Assistant using the Java language. Indeed, the transition from C to C++ is effort consuming, and to switch from C to Java would have consumed the same amount of effort. The Java version would have permitted to have a centralised development of the application Visual Assistant, which would have allowed to get the same run-time of the PC and the Mac versions. Choosing Java, the evolution of the software would have become very easy. If the choice of C++ instead of Java is due to the speed, one can argue that the computers are more and more faster, and that the handicap of Java will sure overcome in the near future.

Let us note that Visual Assistant has also to run on old computers, in order to prevent new investment. But, this fact could also be a handicap in the future.

---

[20] Bruce Eckel, *Thinking in Java*, Prentice Hall, New Jersey, second edition, p.817

2) The architecture of object-oriented components

a) Short description of the important components

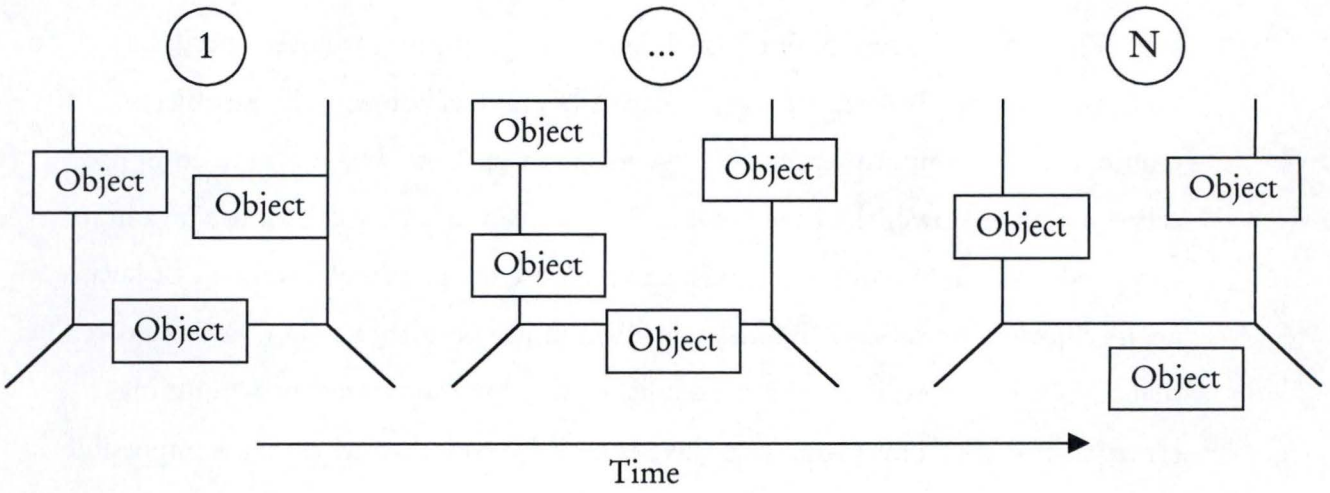


Figure 1: Theatre as a succession of different pictures, as a movie

Deriving from what we have seen earlier, we can say that a play, according to the visual sense, is a succession of object movements in space and time. These objects are within a three-dimensional space, which is the theatrical stage.

We can try to modelise our architecture following what is in the described abstract representation. Evidently, as the Visual Assistant software use is concerned with only the sight sense, we have limited ourselves to the description of the visual aspect of the theatre.

In this definition, one can see three important notions:

- *Physical objects*
- *Three-dimensional space*
- *Passage of time*

As a consequence, we can try to make a correspondence between the mentioned notions with the abstractions of our object-oriented architecture. We have modelised three classes; each one corresponds to an abstraction.

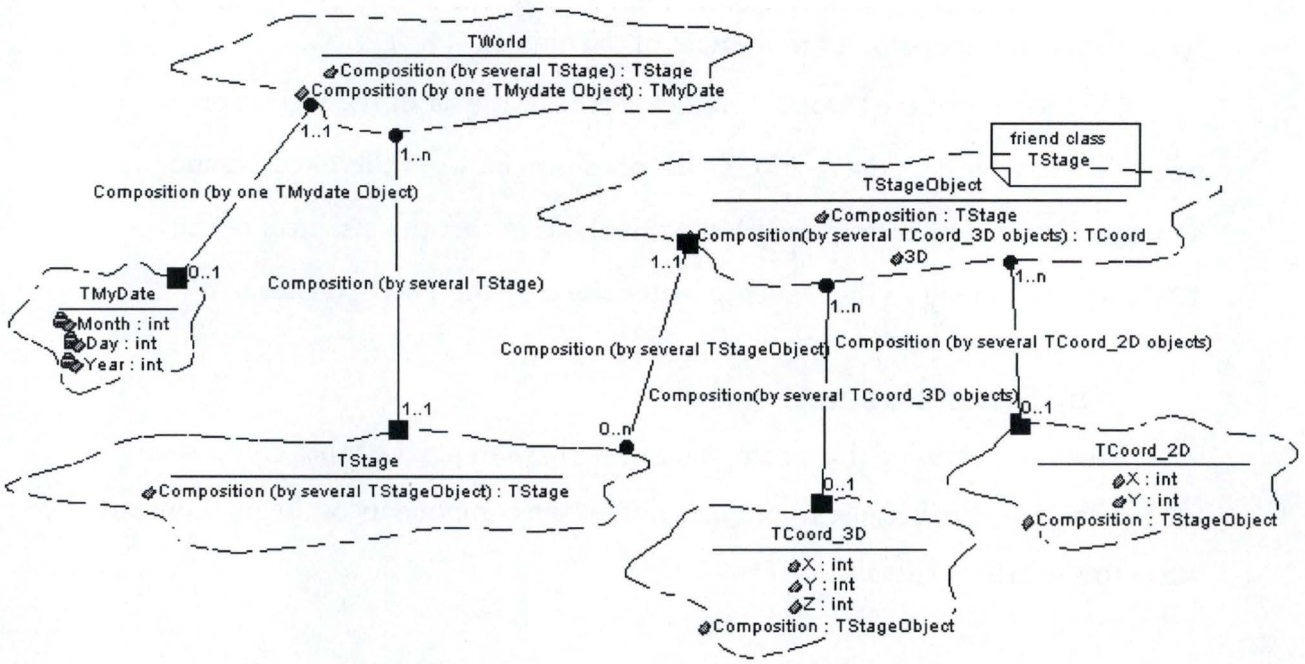


Figure 2: The Three Important Classes (design in Boosh notation)

We have divided the architecture in three principal classes. (see Figure 2)

- `TWorld`
- `TStage`
- `TStageObject`

The *mould* of all the objects corresponding to the *physical objects* of theatre is the `TStageObject` class. Examples for such physical objects are the characters, the circles, etc. These physical objects belong to a theatrical stage, within a *three-dimensional space*. We have modelised this by the means of the `TStage` class. Knowing that the physical objects are contained within the stage, we have used a relation of composition between the `TStageObject` and `TStage` classes. The objects of the first class are data members of the second class. These theatrical stages allow the movement of physical objects, so, a *passage of time*.

In order to modelise this last dimension, we can see a succession of stages that are related to time unit. Each stage has fixed objects, in a position dependent on the time. In other words, we can compare the different `TStage` objects to the pictures of the movies, which are showed $1/24^{th}$ second[21] each. When we run these images, we get an impression of movement of the objects.

We have used the `TWorld` class to represent the set of the succession of stages. We have also used a relation by composition between the `TWorld` and `TStage` classes. The reason underlying this choice is that the instances of the `TStage` class belong to the sequence materialised by the `TWorld` class.

b) Other classes

Before describing these components, we have to see the other components used in the software because the explanation of the components of the previous step refers to the other classes.

On another side, we have defined two other classes, which enable us to capture the two-dimensional and three-dimensional positions of the objects: `TCoord_2D` to keep the position in the two-dimensional space, and `TCoord_3D` to keep the positions in a three-dimensional space.

We have also the predefined classes from *Borland C++ Builder* that we use in the Visual Assistant software. The first is `TList` that enables to store and maintain elements in a list. Of course, this class is very useful to keep the trace of the `TStageObject` in a list, which is in a `TStage` object. This class also allows us to keep trace of the different `TStage` objects in a list, which is in the unique `TWorld` object. Of course, this list is dynamic and only the limits of memory bounds the number of elements.

[21] In the standard movies, we have twenty-four pictures per second.

The class offers these different services:

- Add or delete the objects in the list.
- Rearrange the objects in the list.
- Locate and access objects in the list.
- Sort the objects in the list.

Another class is `Tcanvas`. It provides an abstract drawing space for objects, which must paint their own images. We use `TCanvas` as a drawing surface for objects that draw an image of themselves. Standard window controls such as edit controls or list boxes do not require a canvas, as Windows draws them. We can see the objects from this class as a painting. In fact, all the functions of this class enables us to define the characteristics of a painter: which pen to use, which colour, ... This class allows also to load a picture from a file on a disk or other sources, and it enables to modify the loaded picture. The allowed functionalities are:

- To specify the type of brush, pen to use.
- To draw and fill a variety of shapes and lines.
- To paint graphic images.
- To enable a response to changes in the current image.

The `TBitmap` class contains an internal image of the bitmap graphic and automatically manages its drawing. A bitmap is a powerful graphic object used to create, manipulate (scale, scroll, rotate, and paint), and store images as files on a disk.

To represent our objects from the `TCanvas` class, we must use a `TBitmap` class. In fact, the `TBitmap` class contains the function to display the picture that it has in its `TCanvas` field. In other words, we can create dynamically a `TCanvas` object, but we cannot display it without the use of a `TBitmap` object.

c) Description of the important components

Above, we have the complete architecture that we have shortly described in the previous point. We can explain the different classes with their functions and properties to understand the behaviour of each.[22] (See figure 3)

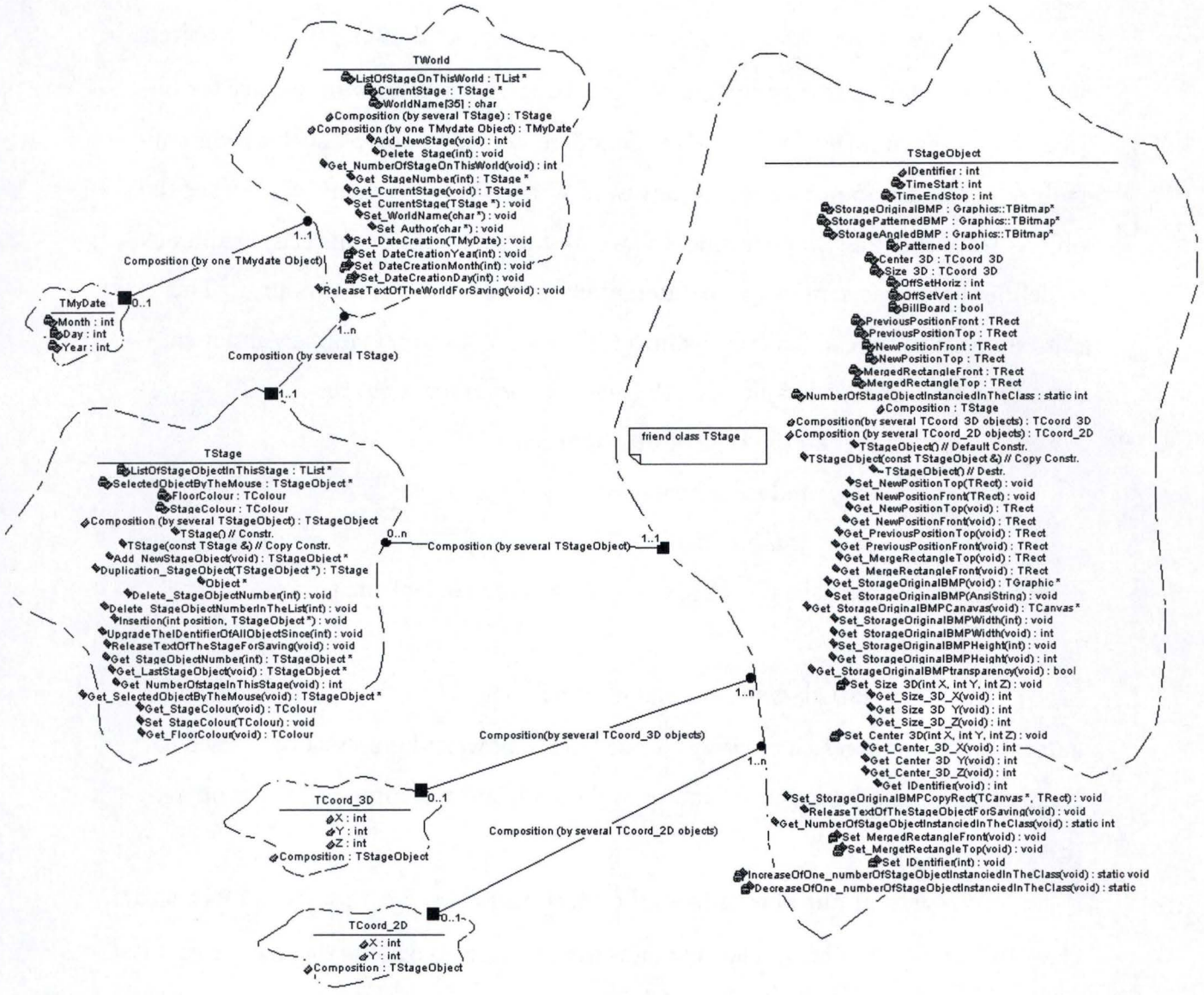


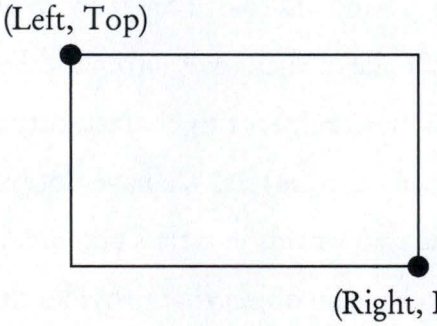
Figure 3: The Three Important Classes with functions and properties (design in Boosh notation)

[22] The complete code of the classes is in Appendix.

- `TStageObject`

The data member the most important is `StorageOriginalBMP`. It enables the capture of the picture of the object in the computer memory. This class is used, not only for drawing objects, but also for picking pictures from a library. `StorageOriginalBMP` is a `TCanvas` object creates dynamically, and it needs to be placed in a `TBitmap` object for its displaying.

There are also the data members to keep the positions 2D of the object: `PreviousPositionFront, PreviousPositionTop, NewPositionFront, NewPositionTop`. All of these data members are a `TRect` type, i.e. a structure keeping the co-ordinates of the upper-left corner and the lower-right corner. In fact, all pictures are captured in a rectangle that represents the area of the object.




```
struct TRect
{ int Left;
  int Top;
  int Right;
  int Bottom;
};
```

Figure 4: Representation of `TRect` type

We can explain the different variables as follows:

- The "`Previous...`" data keep the previous positions when the "`New...`" data change.
- The "`New...`" data keep the new positions.
- The "`Top...`" data keep the positions in the top view, i.e. from the above of the stage.
- And the "`Front...`" data keep the positions of the front view, i.e. the normal sight of the stage for a spectator.

All of these data are generated by the correspondent followed functions: `Set_NewPositionFront`, `Get_NewPositionFront`, `Set_NewPositionTop`, `Get_NewPositionTop`, `Get_PreviousPositionFront`, `Get_PreviousPositionTop`.

We can remark that we do not have at disposal functions to change the "`Previous...`" variables. In fact, we do not need them because the functions managing the "`New...`" data modify automatically the correspondent variable "`Previous...`" to keep the previous value of the "`New...`" variables.

We also have data members to keep the 3D co-ordinates of the centre of the object: `Center_3D`, and another variable to keep the size of the object in relation to this centre: `Size_3D`. We have member functions of the class at one's disposal to change and to get the information. As the 2D and 3D co-ordinates are in closer relations, it is important that when we modify one of them, we automatically modify the other. In fact, all the programmed functions respect this characteristic. In this manner, we never confuse the two kinds of co-ordinates. We have not put the two functions for the conversion between the two worlds in a class because, if we do so, we instantiate the class just one time to have an object that provides the conversion functions. But, we can do that if we want to keep the two functions in a centralised place. In the Visual Assistant software, we have put the global variables in a class to have this sort of variables in one place, but we know that we instanciate the class just one time during an execution of the software.

We have the `TimeStart` and `TimeStop` data members to enable us to manage the succession of our different stages like a movie. These two data define respectively the beginning and the end of the moment of the apparition of the stage.

Finally, this class is characterised by another functionality. It is '`friend`' of the `TStage` class. By the definition of `friend`, the functions of the `TStage` class can modify the data members of the `TStageObject` class. This functionality decreases the encapsulation, but does not suppress all the security because it is the class itself that declares its friends.

As we have seen, this class is a good abstraction to manage and to represent the objects of our software. It is the *mould* of these objects, and we can create our new objects by instantiating the class whenever we want to have all the objects we wish. The management of the object is achieved by the different functions that the objects take from their mould.

- `TStage`

Now, our objects need a stage to stay. We propose another abstraction to manage the objects on a stage: the `TStage` class.

As we have said, this class is the abstraction of the stage in our theatre. But this is not really true. In fact, it is a theatre stage without the fourth dimension, in other words, that is a theatre stage with the objects and the actors without movement. Of course, we can move, and change the objects properties, but at the end of our modification, the stage is static. If we want to make some movement, we would need some different stages and simulate the moving of the things in our theatre.

The data members represent all things what we have on our stage theatre: a set of the objets and actors `ListOfStageObjectInThisTheatre`, the stage colour and the floor colour. Moreover, we have another data member, which is the selected object or actor on which we want to apply a function as *move deep*, *change colour*...

Of course, all these data in the private part are hidden to all other classes, and we have a lot of functions in the `TStage` to manage these private data members.

First, we have the functions to supervise the `TStageObject`. The `Add_NewStageObject` function allows us to add a new `TStageObject` with the default values for its data member. In fact, the new `TStageObject` is placed on the centre of the screen. If we have already the same object on the centre, the constructor of the `TStageObject` changes automatically the position by doing a small shift to the bottom-right to inform the user that (s)he already has one object on screen.

Another one is the `Duplicate_StageObject` with which we can copy an object. The new created object has the same properties as those of the copied object, but the constructor changes automatically the position of the new object, in relation with the copied object. The aim of this change of position is to inform the user that (s)he has two objects on the screen now, by the means of a ting shift between the object to copy and the copied object.

We have two functions to change the colours: `Set_StageColour` for the walls and `Set_FloorColour` for the floor of the theatre stage.

The next functions are the "`Get_...`" functions. We have three functions of this sort for the `TStageObject`, that allow us respectively: to get the current `TStageObject` selected by the mouse, to get the object specified by its identifier, to get the last inserted `TStageObject` in the list. Two other 'Get_...' functions concern the colours: `Get_StageColour` for the walls and `Get_FloorColour` for the floor of the theatre stage.

We also have the functions to delete the `TstageObject`. Two functions are used: one `Delete_StageObjectNumber` to remove a `TStageobject` of the list, and the other `Delete_StageObjectNumberInTheList` to delete a `TStageobject` by removing it of the list and of the memory.

Finally, we have a function `ReleaseTextOfTheStageForSaving` to save the stage theatre in a file. This function releases a specific script with all the characteristics of the stage theatre. Of course, this function automatically calls the function `ReleaseTextOfTheStageObjectForSaving` for all `TStageObject` in it. In this manner, this last function generates the text for all the `TStageobject`, and the `ReleaseTextOfTheStageForSaving` function of the stage pastes these scripts in order to complete its own script. At the end, the function must save all scripts of the stage theatre.

Now, we need an object to get together the different `TStage` to simulate the movement of the `TStageObject` on our virtual stage theatre. This is achieved by the `TWorld` class, which is the abstraction to represent the dynamics between

the different `TStage` elements, just like the movies where we have a lot of different pictures allowing the movements of objects.

- `TWorld`

This abstraction manages the different static theatre stages moulded by the previous class. Of course, we have just one instance at a time in the Visual Assistant software. In fact, we can do more than one work at the same moment.

We have three data members in the private part: one `ListOfStageOnThisWorld` to collect the different `TStage` in a sequence, a pointer on the current `TStage` on the screen to know which `TStage` we want to manage by the means of the operations, and a string to capture the name of the work.

When we instantiate our world, the constructor automatically adds a default `TStage` object, because there is no sense to have a world without at least one theatre stage, by calling the `Add_NewStage` function which is the function to add a new theatre stage with no `TStageObject`.

We may have some other different functions. We have the "Set_..." functions. One `Set_CurrentStage` to show the specified `TStage` on the screen which is considered as the current stage. Another one `Set_DateCreation` to change or to fill the creation date of our work. Another one `Set_Author` to seize the author's name. And the last one `Set_WorldName` to fill the name of our work.

Some reverse functions, called "Get_...", allow us to read the different characteristics set by the "`Set_...`" functions. `Get_CurrentStage` returns the current stage that we see on the screen, `Get_StageNumber` returns the specified stage by its identifier, which is the number in the dynamic sequence of stages.

We also have a different "`Get_...`" function: the `Get_NumberOfStageObjectOnThisWorld` is just a function to return the number of stages in our work.

When we want to delete a `TStage`, we use the `Delete_Stage` function to delete all the `TStageObject` on the stage. Therefore, the function automatically calls the corresponding `Delete_StageObject` function of the `TStage` to delete. When it is done, the function deletes the `TStage`.

Finally, we have a function `ReleaseTextOfTheWorldForSaving` to save all the stages in a file. This function releases a specific script with the all characteristics of the different stages of our world. This script includes all the objects and actors placed on the different stages of interest. Of course, this function automatically calls the function `ReleaseTextOfTheStageForSaving` for all `TStage` in it, and all the stages in the world call the function `ReleaseTextOfTheStageObjectForSaving` for each `TStageobject`. So, this last function makes the text for all the `TStage`, and the `ReleaseTextOfTheWorldForSaving` function of the world pastes these scripts in the way to complete its own script. At the end, the function must save all scripts of our world.

3) How can we design with event system?

The Visual Assistant software runs on an event system. All events are owned by the operating system, and are associated to an application and a function[23], which is executed when the event arrives. The events are generated by the mouse, the keyboard, the operating system, the internal clock, etc. When an event is produced, the operating system places it in a FIFO[24] queue, and controls the queue with regard to the arrival order of the events. Quickly, the system dispatches the events from the list to the corresponding application and the associated function is executed.

In our case, we have three important events: `ButtonMouseDown`, `MouseMove`, and `ButtonMouseUp`. The associated functions are as follows: (See Table 2)

Events	Associated Functions
`ButtonMouseDown`	`StageMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y)`
`MouseMove`	`StageMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)`
`ButtonMouseUp`	`StageMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y)`

Table 2: Associated Functions to Events

All these signatures of the functions are specified by the *C++ Builder*, and we cannot change them. In fact, the system cannot dispatch the events if it does not know where is the corresponding function.

The `ButtonMouseDown` event is used to implement any special processing that should occur as the result of pressing a mouse button. It can respond to left, right, or centre mouse button presses and shift key plus mouse-button combinations. Shift keys are the *Shift*, *Ctrl*, and *Alt* keys. *X* and *Y* are the pixel co-ordinates of the mouse pointer in the *Sender* object, i.e. in our case the drawn stage on our screen.

---

[23] Or script in some other languages.

[24] First In First Out, i.e. we respect the priority of coming.

The `MouseMove` event is used for something happening when the mouse pointer moves within the control. The parameters are identical to those of the previous function.

The `ButtonMouseUp` is utilised to implement special processing when the user releases a mouse button. The parameters are identical to those of the first function.

Now that we have explained a few concepts of the event programming, we can see how Visual Assistant software runs internally. (See figure 5 on the next page)

To allow the communication between the events, i.e. for example to enable the `MouseMove` to know on which object we have clicked before and to generate the event `MouseButtonDown`, we use some different global variables stored in a class. We have no choice, there are no other means to allow the communication between events: all functions can change the variables and we have no security. On another hand, with the class we have centralised the code for the global variables. The name of this kind of global variables for the three events is materialised by the prefix "`Can...`" (See figure 5). Let us briefly describe the features of our figure that we have numbered in sequence:

- We choose the tool we want to use in the toolbox or in the menu.
- After that, we click on the stage[25]. We have two cases. We have chosen an operation, which needs the `MouseMove` and the `ButtonMouseUp` events: *PencilFreeHand*, *Line*, *MoveonPlan*, *MoveInDeep*. Or even, we have chosen an operation which needs just a click and can work endlessly.[26]
- We need to use the event `MouseMove` for the four specified operations in the previous point. In case of the moves, we change the positions of the shape and in the case of the drawing, we draw like when we use a pen.
- In this last event, we finish the work of the four operations: in the case of the move, we apply the new position, and in the case of drawing we store the new shape in the list of stage objects.

---

[25] Thus, the mouse button is down.

[26] i.e. all other tools than those described in the previous case.

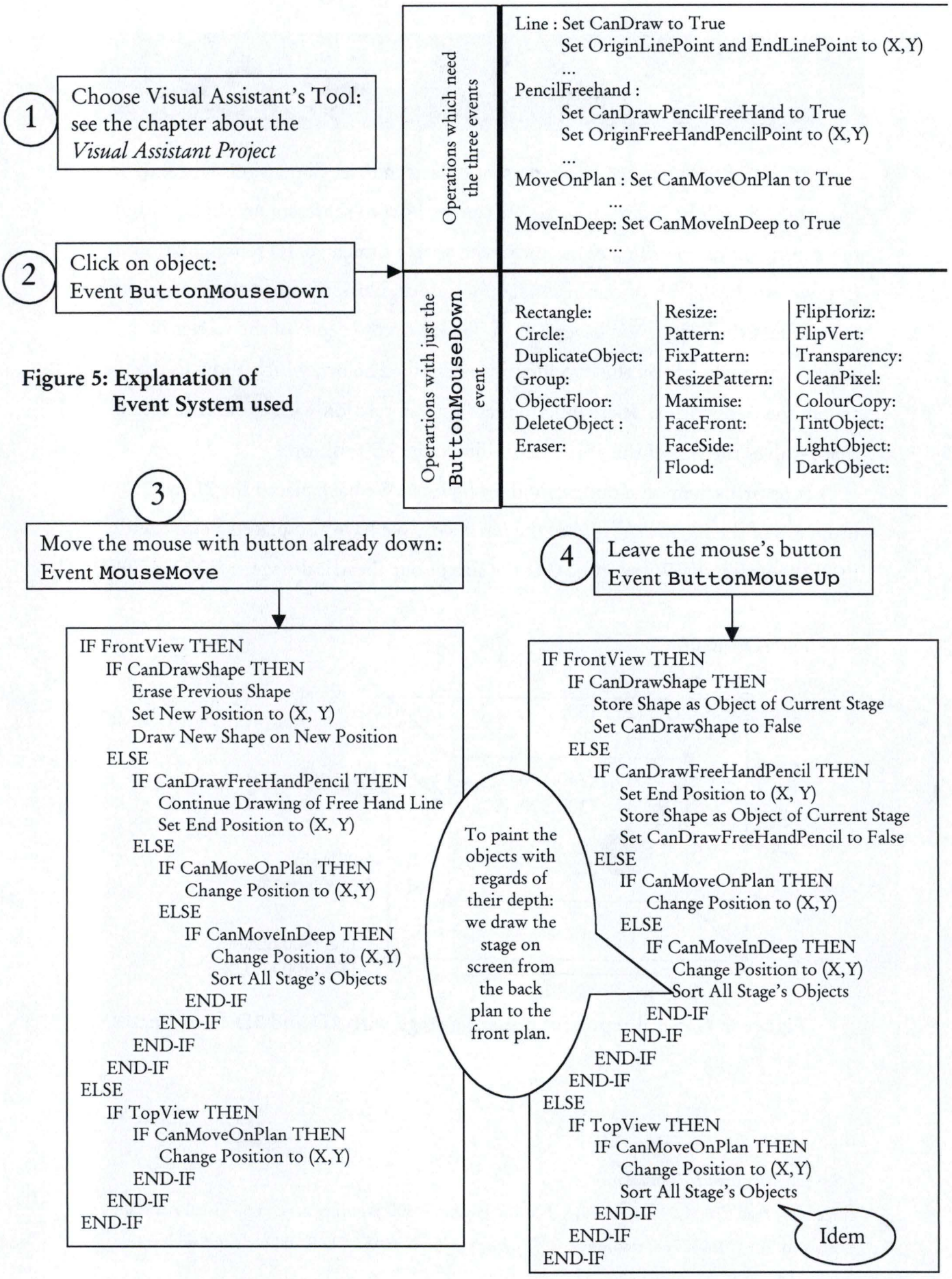


Figure 5: Explanation of Event System used

4) How can we draw a three-dimensional object on a two-dimensional screen?

The aim of the software is to represent objects in three dimensions whereas, on our screen, it will be bidimensional. We can see how to represent an object with its three dimensions co-ordinates on our screen with a simulated 3D representation. In fact, we use the default of our vision, and we use an illusion with a *vanishing point*[27] to represent the object. We have such an illusion on the depth of the screen. We achieve this principle by the two functions described below, which allow us to switch the two worlds. These functions are used in relation with another function that resolves the size of the object depending of its 3D position.

Let us first schematise our method. In figure 6, We have placed the 2D co-ordinates of the *stage back*[27] and of the *forestage*[27]. We have also placed a circle with its 2D co-ordinates. We suppose that the size of our theatrical stage is (800 X 600)[28].

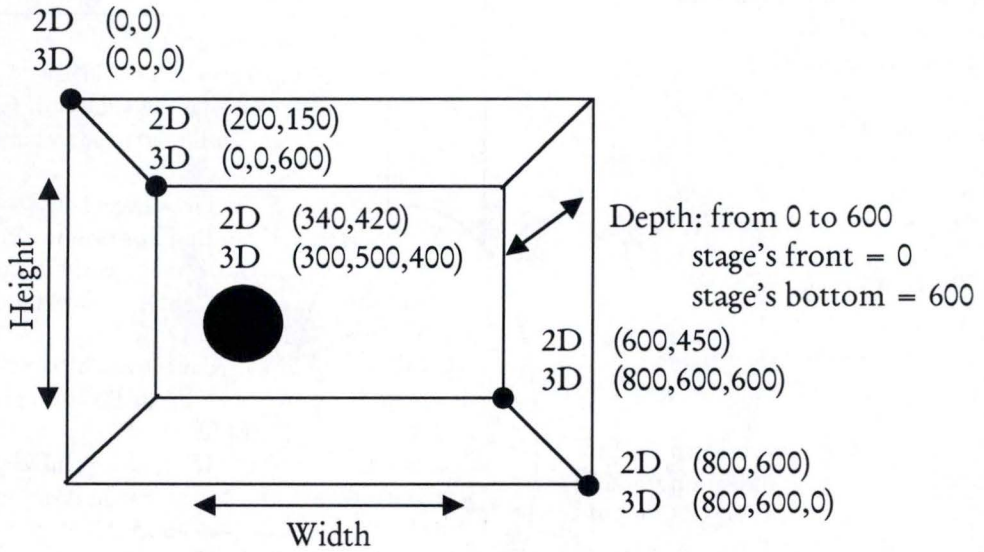


Figure 6: General representation of a stage with 2D and 3D co-ordinates

---

[27] See appendix *French Translator*

[28] We must read (800 X 600) as Width = 800 and Height = 600. Another version of Visual Assistant exists with the (1024 X 768) resolution. This latter version is on the CD-ROM provided with this work.

a) 3D World to 2D World

```
int Convert3Dto2D(int MiddleLine, int 3D_Coord, int Deep)
{ int D1 = VfTB.Get_UserToScreenDist() + Deep;
  int D2 = VfTB.Get_UserToScreenDist();
  float H1 = 3D_Coord - MiddleLine;
  float H2 = (H1 / D1) * D2;

  return (MiddleLine + H2);
}
```

The function `Convert3Dto2D(...)` allows us to switch from the 3D world to the 2D World. In fact, we use the function in two steps: first with the X co-ordinates, and then with the Y co-ordinates. We also give the stage depth to solve the X and Y co-ordinates in 2D. Moreover, if we calculate the 2D X co-ordinate, we need the middle of stage width, and in the case of the 2D Y co-ordinate, we need the middle of stage height. The `VfTB.Get_UserToScreenDist()` allows us to have a distance between the user and the screen. In fact, the vanishing point and the perceptive are calculated according the distance.

We have in the three tables (see Table 3, Table 4 and Table 5 on the next page) of the example the calculation of our previous stage: the 2D positions of the back stage, the 2D positions of the centre of the circle.

The columns of these tables must be read as follows: the `MiddleLine` is the middle of the corresponding dimension as explained before, `3D_Coord` is the 3D X or 3D Y position, the `Depth` is the 3D Z position, and 'Result' is the returned value of the function, i.e. the 2D X or 2D Y position. All the results are obtained for a distance equal to 600 (600 is the distance between the screen and the user).

Table 3: we can solve the position of the back stage. Two first lines are for the co-ordinates of the upper left corner of the rectangle and the other lines are for the co-ordinates of the lower right corner of the rectangle.

	MiddleLine	3D_Coord	Depth		Result
Corner Top Left X	400	0	600	=	200
Corner Top Left Y	300	0	600	=	150
Corner Bottom Right X	400	800	600	=	600
Corner Bottom Right Y	300	600	600	=	450

Table 3: Position of the Back Stage

Table 4: we can solve the position of the front stage. We need four calls of the function. Two for the co-ordinates of the upper left corner of the rectangle that we can read in the two first lines and two for the co-ordinates of the lower right corner of the rectangle that we can read in the two last lines of the table.

	MiddleLine	3D_Coord	Depth		Result
Corner Top Left X	400	0	0	=	0
Corner Top Left Y	300	0	0	=	0
Corner Bottom Right X	400	800	0	=	800
Corner Bottom Right Y	300	600	0	=	600

Table 4: Position of the Front Stage

Table 5: we can solve the position of the centre of the circle. We need to use the function twice. The first line is for the X co-ordinate and the last for the Y co-ordinate.

	MiddleLine	3D_Coord	Depth		Result
Center 2D X	400	300	400	=	340
Center 2D Y	300	500	400	=	420

Table 5: Position of the Centre of the Circle

We can now write a new function for solving in one step the 2D X and Y co-ordinates:

```
TCoord_2D Convertion3DTo2D(int MiddleWidth,
                           int MiddleHeight,
                           int X, int Y, int Z)
{ TCoord_2D.X = Convert3Dto2D(MiddleWidth, X, Z);
  TCoord_2D.Y = Convert3Dto2D(MiddleHeight, Y, Z);

  return TCoord_2D;
}
```

b) 2D World to 3D World

```
int Convert2Dto3D(int MiddleLine, int 2D_Coord, int Deep)
{ int D1 = VfTB.Get_UserToScreenDist();
  int D2 = VfTB.Get_UserToScreenDist() + Deep;
  float H1 = 2D_Coord - MiddleLine;
  float H2 = (H1 / D1) * D2;

  return (MiddleLine + H2);
}
```

This function allows us to switch from the 2D world to the 3D World. As the previous function, we use the function in two steps: first with the X co-ordinate, and then with the Y co-ordinate. We need the depth being fixed. We need the middle of the corresponding dimension. We can see that we also need the depth of the object, i.e. the third dimension. In fact, we need to known the value of the depth, when we use the function for the conversion from the 3D to the 2D because a same position in 2D can give a set of position along a line in 3D, and we have to find the correct position on this line.

Let us use just one table explaining the function: the 3D positions of the centre of the previous circle. The columns of the table must be read as follows: the `MiddleLine` is the middle of the corresponding dimension, `2D_Coord` is the 2D X or 2D Y position, the `Depth` is the 3D Z position, and `Result` is the returned value by the function, i.e. the 3D X or 3D Y position.

Table 6: We can solve the position of the centre of the circle by using the function twice. The first line is for the X co-ordinate and the last for the Y co-ordinate.

	MiddleLine	2D_Coord	Depth		Result
Center 3D X	400	340	400	=	300
Center 3D Y	300	420	400	=	500

Table 6: Position of the Centre of the Circle.

Of course, we could have written another function for solving in one function call the 3D X and Y co-ordinates:

```
TCoord_3D Convertion2DTo3D(int MiddleWidth,
                           int MiddleHeight,
                           int X, int Y, int Z)
{ TCoord_3D.X = Convert2Dto3D(MiddleWidth, X, Z);
  TCoord_3D.Y = Convert2Dto3D(MiddleHeight, Y, Z);
  TCoord_3D.Z = Z;

  return TCoord_3D;
}
```

5) How can we save?

We have opted for a standard ASCII file and own representation of the Visual Assistant Works. The advantage is that we can easily use a lexer[29] to read the generated files. We can see below the BNF[30] of the saved files that describes the content of files and stages:

---

[29] A lexer can analyse a text to find same patterns. The normal use of the lexer is to generate a program.

[30] Backus-Naur Form

```
<VA> ::- [Visual Assistant 1.0]<World>

<World> ::- [World]{WorldName}<String>{Author}<String>
            {DateCreation}Month:<Integer>/Day:<Integer>
            /Year:<Integer><ListStage>

// We cannot have more than one 'World' by Visual Assistant file

<ListStage> ::- <Stage> | <Stage><ListStage>

<Stage> ::- <StageProperties> | <StageProperties><ListStageObject>

<StageProperties> ::- [Stage]{FloorColour}<Integer>
                      {StageColour}<Integer>

<ListStageObject> ::- <StageObject> | <StageObject><ListStageObject>

<StageObject> ::- <empty> | [StageObject]{TimeStart}<Integer>
                 {TimeStop}<Integer>{StorageOriginalBMP}<BMP>
                 {StoragePatternedBMP}<BMP>{StorageAngledBMP}<BMP>
                 {Patterned}<Boolean>{Center_3D}<Coord_3D>
                 {Size_3D}<Coord_3D>{OffSetHoriz}<Integer>
                 {OffSetVert}<Integer>{BillBoard}<Boolean>
                 {PreviousPositionFront}<Rectangle>
                 {PreviousPositionTop}<Rectangle>
                 {NextPositionFront}<Rectangle>
                 {NextPositionTop}<Rectangle>
                 {MergedRectangleFront}<Rectangle>
                 {MergedRectangleTop}<Rectangle>

// empty = we can have 'Stage' without 'StageObject'

<empty> ::- empty
<String> ::- … …
<BMP> ::- … …
<Boolean> ::- True | False

<Integer> ::- <Digit> | <Digit><Integer>
// It is obligatory to have at least one digit !
Digit ::- 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7| 8 | 9

<Coord_3D> ::- <Integer>;<Integer>;<Integer>
<Rectangle> ::- <Integer>;<Integer>;<Integer>;<Integer>
```

The saved file respects the same decomposition that for the classes: *World*, *Stage* and *StageObject*. The ASCII characteristic of the file enables us to modify the file with a text editor. We can see example of a saved Visual Assistant file to explain the power of a lexer[31]:

```
[Visual Assistant 1.0]
[World]{WorldName}Monde{Author}Auteur
       {DateCreation}Month:4/Day:11/Year:2000
[Stage]{FloorColour}45{StageColour}65
[StageObject]{Patterned}False{Center_3D}304;196;0{Size_3D}100;100;0
             {OffSetHoriz}0{OffSetVert}0{BillBoard}False
             {PreviousPositionFront}0;0;0;0
             {PreviousPositionTop}0;0;0;0
             {NewPositionFront}254;146;354;246
             {NewPositionTop}254;386;354;398
             {MergedRectangleFront}0;0;354;246
             {MergedRectangleTop}0;0;354;398
[StageObject]{Patterned}False{Center_3D}135;135;0{Size_3D}161;28;0
             {OffSetHoriz}0{OffSetVert}0{BillBoard}False
             {PreviousPositionFront}0;0;0;0
             {PreviousPositionTop}0;0;0;0
             {NewPositionFront}121;55;149;216
             {NewPositionTop}121;386;149;398
             {MergedRectangleFront}0;0;149;216
             {MergedRectangleTop}0;0;149;398
```

We can see that it is easy to understand the meaning of the saved file. The advantage of this method is that the lexer format looks like the BNF format. We define our grammar in BNF and then we write the lexer file[32]. In the lexer file, we mix the grammar with some little C language code to describe how to save our objects from Visual Assistant. After writing, we execute the lexer software on the lexer file, and it returns a fully C file, which we can introduce in our project. In other words, the lexer software has written the C language code for us.

---

[31] Some carriage returns and tabulations are used to enable the reading easier.

[32] The code of the lexer of Visual Assistant is in the Appendix.

6) How can we move in depth?

We have a 2D screen, and we want to simulate the 3D with the illusion enabled by the vanishing point and the perspective. We have already mentioned the question, but how can we make visible the depth movements on the screen when we are on the front view?

First, we have four different quadrants on our screen, on each we have different directions for the vanishing point. (See Figure 7)

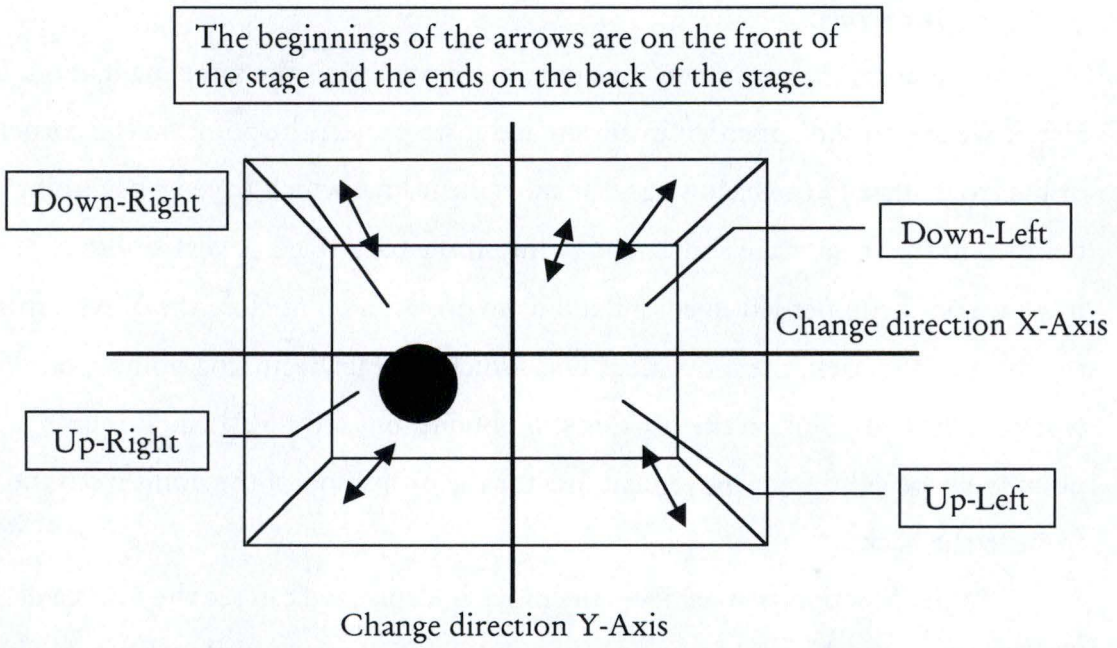


Figure 7: The Four Quadrants, the Vanishing Point, and the depth movements

As we see, the back of the stage is materialised by the illusion of the decreasing in size of the rectangle from the front of the stage. To obtain this result, we need two axes:

- Above the X-Axis, we move from the front to the back by downward on the screen
- Below the X-Axis, we move from the front to the back by upward on the screen
- Left the Y-Axis, we move from the front to the back by moving right on the screen
- Right the Y-Axis, we move from the front to the back by moving left on the screen

Of course, the movement line does not have always the same inclination. In fact, if we are on the upper-left quadrant and if we project the point on the corner of the front stage to the back stage, the movement line, which have its beginning point is on the front stage and its end point on the back stage, is very oblique. But, if we are on the upper-left quadrant and if we project a point close the Y-Axis from the front to the back, the movement line, which have its beginning point is on the front and its end point on the back, is still oblique, but more vertical. For each point we have a different movement line for the projection of the point from the front to the back.

In the function that achieves the move in depth, we can see the four cases. The function calculates, before a movement, on which quadrant the object is. The function uses the conversion function `Convert3Dto2D(...)`, which is described in the point *3D World to 2D World*, in order to have the new position 3D in relation with the 2D position of the pointer of the mouse. After that, the function must resize the object to have illusion of the depth. To do that, it calls a function that enables the resizing of the object in relation with the 3D position.

7) Conclusion

In this chapter we have discussed C++. Firstly, we have recalled the object-oriented programming used by C++. After that, we have seen a method, among a lot of other methods, that we have used to make our PC version of Visual Assistant software.

In fact, we have chosen this method of object-oriented programming to have a good architecture where it is easy to add some new functionalities, to add the remaining functionalities coming from the Mac version, and to manage the future evolution of computers.[33]

Secondly, we have discussed the architecture made in this work. First, we have explained in a general way the classes by linking classes as a representation of reality abstractions. Next, we have seen the classes in depth and we have looked at the different relationships between the classes.

Thirdly, we have illustrated the event system to understand the working of the software. In fact, we are used to procedural softwares, but the event programs are very different. Their executions are not executed independently of their external environment, which provides events.

Fourthly, we have explained the more important functions of the software. We have seen the graphical problems, i.e. how to do the transition between the two-dimensional and the three-dimensional worlds and how to move an object in depth with a two-dimensional representation on the screen. Moreover, we have seen how to use a parser to read the saved files, which contains the work we have made previously with the software on the same or another machine.

---

[33] The different manners used to make the migration of Visual Assistant from the Mac platform to the PC platform are described in the chapter *Reusing the existing knowledge*

Conclusion

We have tried to define some notions that one can encounter in the artistic field such as creativity and improvisation. We have shown that the notion of creativity is open to debate and that no consensus is reached about its origins. We went further and tried to see the different steps involved in this phenomenon i.e. creativity, with the hope to be provided by more clarifications or at least a track that we can follow to understand creativity. Unfortunately, the Poincaré's four phases of creativity apply only scientific creativity and not to the artistic creativity which is our main issue in this work. After adopting one of the definition given to creativity, we have explored whether the computer scientists are themselves creative and how they can capture and modelise the creative activities of theatre artists. We have shown that computer scientists are as creative as any other person in artistic fields or in any other field. We have also shown that although programs are governed by unambiguous rules, they offer a means to artists to express their creativity. We have established that the critics (which say that rules/constraints are opposite to freedom, which is the essence of any artistic activity) are not founded.

Then we have explored the theatre direction and we have tried to understand how does it work. The aim here is to understand the requirements of one of the end users of the software Visual Assistant.

At once we have tried to find out the possibilities and boundaries of the virtual reality when it deals with the artistic field, and to understand and to develop an analysis of the criticism of the virtual reality. Note that this criticism is arisen especially by artists.

We have presented in some detail the software Visual Assistant, and we have analysed whether it meets the requirements of the end users. The software designed in this work is based on a Mac version implemented in C language. This software runs now on a PC platform and was implemented through the use of C++ language.

We have started by exploring different fields dealing with the reuse of software. Then we have presented the architecture of the software and have done a presentation of C++ language and its comparison with C language.

Although the PC version is available on the Internet, we think that a lot of work have to be done before the software Visual Assistant deserves to be used especially by theatre directors.

Bibliography

Artaud Antonin, *The theatre and Its Double*, Caulder and Boyers, London 1970.

Bass L., Clements P. and Kazman R., *Software Architecture in Practice*, Addison-Wesley, 1997.

Baudelaire, Le public moderne et la photographie, in: *Salon*, 1859.

Beardon Colin and Enright Terry, Computers and Improvisation: Using the Visual Assistant for Teaching, *Digital Creativity*, United Kingdom, University of Plymouth, 1999, Vol. 10, No. 3, pp. 153-166.

Beardon Colin and Enright Terry, The Visual Assistant: designing software to support creativity, in: *Proceedings of (Computers in Art & Design Education) CADE'99 Conference*, University of Teesside, 5-7 April 1999. This paper can be found at http://www.esad.plym.ac.uk/personal/C-Beardon/papers/paper9901.html

Beardon Colin and Tuomola Mika, Thematic Network Project: Multimedia Learning for the Theatre, *InterELIA* (European League of Institutes of the Arts), issue three, Winter 1997-1998, pp. 17-22.

Beardon Colin, Creative practices and the design of virtual environments, United Kingdom, Exeter School of Arts and Design. This paper can be found at http://www.esad.plym.ac.uk/VA/papers/2001.html

Beardon Colin, Digital creativity in the 21st Century, in: *Proceedings of ICFAD '99.*

Beardon Colin, Gollifer Sue, Rose Christopher, Worden Suzette, Computer Use by Artists & Designers, in: M. Kyng and L. Mathiassen (Eds.) *Computers and Design in Context*, MIT Press, Cambridge (Mass.), 1997, pp. 27-50. This paper can be found at http://www.esad.plym.ac.uk/personal/C-Beardon/papers/9510.html under the title *Designers as users.*

Beardon Colin, Multimedia Learning Tools for the Theatre, 1998. Paper presented at the MESH Conference, London 1-2 October 1997. This paper can be found at http://www.esad.plym.ac.uk/personal/C-Beardon/papers/9701.html

Beardon Colin, The design of software to support creative practice, 1999. Paper presented at the *International Conference on Design and Technology Educational Research and Curriculum Development (IDATER 99)*, University of Loughborough, 23-25 August

1999.
This paper can be found at
http://www.esad.plym.ac.uk/personal/C-Beardon/papers/paper9902.html

Beardon Colin, *The Visual Assistant website at*:
http://www.esad.plym.ac.uk/VA/index.html
http://www.esad.plym.ac.uk/projects/VA.html

Beardon Colin, The Visual Assistant: institutional perspectives on the development of software, United Kingdom, Exeter School of Arts and Design. This paper can be found at
http://www.esad.plym.ac.uk/VA/papers/9701.html

Beau Frank, Dubois Philippe, Leblanc Gérard, *Cinéma et dernières technologies*, De Boek, Bruxelles 1998.

Becq de Fouquières Louis, *L'Art de la mise en scène : essai d'esthétique théâtrale*, Entrevues, Marseille 1998, reediting of 1884.

Blair D. and Mayer T., Tools for an Interactive Virtual Cinema. in: Trappl, R. and Petta, P (Eds.) *Creating Personalities for Synthetic Actors: towards autonomous personality agents*, Springer, Berlin, pp. 83-91.

Bodart F. and Pigneur Y., *Conception assistée des systèmes d'information : Méthode – Modèles – Outils*, Masson, Paris 1989, second edition.

Boden Margaret A., *The creative mind: myths and mechanisms*, Abacus, London 1992.

Boden Margaret A., What is creativity?, in: Boden, Margaret A. et al., *Dimensions of creativity*, MIT Press, Cambridge (Mass.), 1994.

Bott F. and Ratcliffe M., Reuse and design, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., Chapman & Hall, London 1992, pp. 35-51

British Actors' Register website at:
http://www.internet-ireland.ie/power/actor/actor.htm

Burrows David, Visual Assistant, *OutLine*, Wimbledon School of Art, Theatre Department, issue 8, autumn 1999.

CADE 99 Conference at:
http://www.tees.ac.uk/CADE99/

Carrière Jeromy S. and Kazman Rick, *The Perils of Reconstructing Architectures*, ACM, Carnegie Mellon University, Pittsburgh (Florida), 1998.

Clarinval André, *Le language C*, Institute S[t]-Laurent Sup', September 1994.

Computers in Art & Design Education (CADE) website at:
http://esad.plym.ac.uk/CADE/

Crary J., *Technique of the Observer. On Vision and Modernity in the Nineteenth Century*, Cambridge, Mass., MIT Press, October 1990.

Daisy's Amazing Discoveries website at:
http://www.coronet.fi/daisy/

Digital Creativity journal website at:
http://www.swets.nl/sps/journals/dc1.html

Director's Assistant website at:
http://vconf.hut.fi/hamlet/

Distributed Video Production website at:
http://viswiz.gmd.de/DVP/Public/deliv/deliv.413/deliv.413.html#VA

Dramatic Exchange website at:
http://www.dramex.org/

Eckel Bruce, *Thinking in C: Foundations for Java & C++*, Prentice Hall, New Jersey

Eckel Bruce, *Thinking in C++ : Standard Libraries & Advanced Topics*, Prentice Hall, New Jersey,Vol. 2, second edition, 2000.

Eckel Bruce, *Thinking in C++*, Prentice Hall, New Jersey, Vol. 1, Second Edition, 13 January 2000.

Eckel Bruce, *Thinking in Java*, Prentice Hall, New Jersey, first edition.

Eckel Bruce, *Thinking in Java*, Prentice Hall, New Jersey, second edition.

ELAC on-line Plays website at:
http://www.perspicacity.com/elactheatre/index.html

Fick Jean-Marc, Cybergoût et sensorialité numérique, in: *Champs Visuels n°5*, L'Harmattan, Paris 1997.

Forst A. and Yarrow R., *Improvisation in Drama*, Macmillan, London 1990.

Frazer A., Reverse engineering - hype, hope, or here?, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., *UNICOM Applied Information Technology*, Chapman & Hall, Vol. 12, London 1992, pp. 209-243.

George Coates' performance works website at:
http://www.georgecoates.org/

Goldsack S.J., Re-engineering software for distributed execution, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., *UNICOM Applied Information Technology*, Chapman & Hall, Vol. 12, London 1992, pp. 357-386.

Hall P.A.V., Editorial Introduction, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., *UNICOM Applied Information Technology*, Chapman & Hall, Vol. 12, London 1992, pp. xiii-xvii.

Hall P.A.V., Software reuse, reverse engineering, and re-engineering, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., *UNICOM Applied Information Technology*, Chapman & Hall, Vol. 12, London 1992, pp. 3-32.

Hildesheimer W., *Mozart*, London 1983.

Hodgson R., The impact of software reuse on object-oriented methods, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., *UNICOM Applied Information Technology*, Chapman & Hall, Vol. 12, London 1992, pp. 159-205.

Holloway S., Re-engineering business systems to use the next generation of software, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., *UNICOM Applied Information Technology*, Chapman & Hall, Vol. 12, London 1992, pp. 271-282.

Huizinger J., *Homo Ludens*, Beacon Press, New-York 1955.

Jones R., How applicable is the object-oriented approach to the IS environment?, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., *UNICOM Applied Information Technology*, Chapman & Hall, Vol. 12, London 1992, pp. 137-157.

Juvet André, *Réflexions de comédien*, Sablon, Bruxelles - Paris 1944.

Koestler A., *The Act of Creation*, Picador, London 1975.

Kruzela I. and Brorsson M., *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., *UNICOM Applied Information Technology*, Chapman & Hall, Vol. 12, London 1992, pp. 521-534.

Laurel B., *The computer as Theatre*, Academic Press, 1990.

Livingston Lowes, *The Road to Xanadu: A study in the ways of the Imagination*, Constable, London 1951.

McCullough Malcolm, *Abstracting craft : The practiced digital hand*, MIT Press, Cambridge (Mass.), 1998 for the paperback (First published in 1996).

McGill R., Reverse engineering – not yet?, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., *UNICOM Applied Information Technology*, Chapman & Hall, Vol. 12, London 1992, pp. 245-252.

Miche Frédéric, *Computer Sketching: How Software Tools Can Enhance Human Creativity?*, Master's degree dissertation, Institute of Computer Science (FUNDP), Namur 1999.

Müller Hausi A., Reverse Engineering Strategies for Software Migration, *ACM*, (?Canada?), University of Victoria, 1997, p. 660.

Müller Robert C., "Enhancing creativity, innovation and co-operation", in AI & Society (the journal of human-centred systems and machine intelligence), Volume 7 Number 1, Springer-Verlag, London 1993.

Munro M., Software maintenance, reuse and reverse engineering, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., *UNICOM Applied Information Technology*, Chapman & Hall, Vol. 12, London 1992, pp. 573-584

Oualline Steve, *Practical C++ Programming*, O'Reilly, USA, January 1997.

Paxson Vern, *Flex*, March 1995.

Philippe Coiffet, *Mondes imaginaires*, Hermès, Paris 1995.

Poincaré Henri, *The foundations of sciences: science and hypothesis, The Value of Science*, Science and Method, Washington 1982.

Prechelt Lutz, *Technical opinion: comparing Java vs. C/C++ efficiency differences to interpersonal differences*, *Communications of the ACM*, Vol. 42, No. 10, October 1999, pp. 109-112.

Quéau Philippe, *Le virtuel : vertus et vertiges*, collection Milieux, editions Champ Vallon/I.N.A., Seyssel 1993.

Rezvani Serge, *Théâtre: Dernier Refuge de l'Imprévisible Poétique*, collection Apprendre, Actes Sud-Papiers, June 2000.

Rheingold H., *La réalité virtuelle*, Dunod, Paris 1993.

Richard Finkelstein's Designs website at:
http://www.artsozoo.org/rf/

Ryle G., *The concept of mind*, Hutchinson, London 1949.

Satir Gregory and Brown Doug, *C++: The Core Language*, O'Reilly, USA 1996.

Screenwriters and Playwrights website at:
http://www.teleport.com/~cdeemer/scrwriter.htm

Shakespeare Archive website at the Massachusetts Institute of Technology website at:
http://the-tech.mit.edu/Shakespeare/

Shakespearean websites at:
http://www.shakespeare.com
http://www.rdg.ac.uk/AcaDepts/In/Globe/home.html
http://www.gh.cs.su.oz.au/~matty/Shakespeare/

Somerville Ian, *Software engineering*, Addison-Wesley, ?, 19??.

Spolin V., *Improvisation for the theatre*, Pitman Publishing, London 1963

Stroustrup Bjarne, The C++ Programming Language, Addison-Wesley, New Jersey, 1991, second edition

Talent Source website at:
http://www.talent.com.au/ts/menu.html

Theatre Central website at:
http://www.theatre-central.com/

Theatre Department of Wimbledon School of Art website at:
http://www.wimbledon.ac.uk/thatre/techarts/index.htm

Umar Amjad, Application (Re)Engineering : Building Web-Based Applications and dealing with Legacies, Prentice Hall, Piscataway (New Jersey) 1997.

Walton P., The management of reuse, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., *UNICOM Applied Information Technology*, Chapman & Hall, Vol. 12, London 1992, pp. 505-520.

Warden R., Re-engineering - a practical methodology with commercial applications, *Software Reuse and Reverse Engineering in Practice*, edited by Hall P.A.V., *UNICOM Applied Information Technology*, Chapman & Hall, Vol. 12, London 1992, pp. 283-305.

Winograd Terry, Bennett John, De Young Laura, Hartfield Bradley et al., *Bringing design to software*, ACM Press, Baltimore (Md.), 1996.

Appendix

1) FRENCH TRANSLATOR

2) THE CODE OF THE CLASSES

3) THE LEXER CODE

1) French Translator

English	Français
Actor	Acteur
Apron (stage)	Avant-scène
Audience	Spectateurs
Back of the stage	Dernier plan de la scène
Back stage	Arrière-scène
Bottom stage	Plancher de scène
Brainwave	Idée, trait génie, idée lumineuse
Casting	Théâtre : Distribution des rôles, casting Informatique : Conversion explicite de type
(to) Change of scene	Changer de décor
Character	Personnage
Comedian	Comédien
Computer graphics	Infographie
Conductor	Chef d'orchestre
Costume	Costume
Costum(i)er	Costumier
Craft	Manuel, artisanal
Craft document	Description précise, prescriptions de mise en scènes
(to) Direct	Théâtre, Cinéma : Mettre en scène une pièce, un film Théâtre, Cinéma : diriger un film, des acteurs, etc
Direction	Théâtre : Mise en scène (en Pratique et en Abstraction) Cinéma, TV, Radio : Réalisation

English	Français
Director	Théâtre : Metteur en scène Cinéma, TV, Radio : Réalisateur
Drama	Art dramatique Le théâtre
Drama critic	Critique dramatique
Drama person	Personne du drame
Drama work	Œuvre dramatique
Dramatic criticism	Critique dramatique
Dramatis person(ae)	Personnage(s)
Dramatist	Dramaturge
Dress	Costume
Dress rehearsal	Répétition générale
Emotion	Émotion
Essence of the (drama) work	Intentionnalité du texte, de l'œuvre
Feeling	Émotion Sensation
Floor stage	Plancher de la scène
Footlights	Rampe (de lumière) de l'avant-scène
Forestage	Avant-scène
Front of the stage	Premier plan de la scène
Front stage	Avant-scène
Ideas (wo)man	Concepteur
Live theatre	Théâtre de rue
Lighting technician	Éclairagiste
Model	Maquette
Narrative	Récit, narration
(to) Perform	Jouer, représenter une pièce
Performance	Théâtre : Représentation Cinéma : Séance

English	Français
	Musique : Interprétation
Performer	Théâtre :, Acteur Musique : Exécutant, interprète, artiste
Placing in space	Mise en espace, spatialisation
Play	Pièce (de théâtre)
Playwright	Dramaturge
Poetry	Poésie
Practitioner	Spécialiste, expert
Producer	Théâtre : Metteur en scène Cinéma, TV, Radio : Producteur
Production	Théâtre : Mise en scène Cinéma, TV, Radio : Production
Prompter	Souffleur
Rehearsal	Répétition (théâtrale)
Representation	Théâtre : Interprétation (de rôles) Peinture : représentation
(to) Run	Diriger, gérer un théâtre
Scenario	Théâtre : Scénario
Scene	Scène (en Pratique et en Abstraction) Décor
Scenecraft	Scénographie
Scenery	Décors
Scenographer	Scénographe
Screenplay	Cinéma : Scénario
Screenwriters	Scénariste
Setting up and installation	Aménagement
Silent play	Pièce muette
Sketch	Art : Esquisse, dessin à grand trait Théâtre, TV : Sketch

English	Français
(to) Sketch	Esquisser, dessiner à grands traits
Sketching	Dessin à main levée
Specifications	Cahier des charges Description précise, prescriptions
Stage	Scène (en Pratique)
Stage acting	Jeu de scène
Stage design	Décoration théâtrale
Stage designer	Décorateur de théâtre
(Stage) direction	Direction scénique
(Stage) director	Metteur en scène
(to) Stage-manage	Mettre en scène
Stage manager	Régisseur (d'une pièce)
Stage painter	Peintre de décors
(Stage) play	Pièce de théâtre
Stage playing	Jeu de scène
Stagecraft	Technique de la scène
Stagehand	Machiniste
Staging	Mise en scène (en Pratique)
Theatre hall	Salle théâtrale
Theatre manager	Directeur de théâtre
Theatre production	Mise en scène théâtrale
Theatrical art	Art théâtral
Vanishing point	Point de fuite
Walk-on (part)	Rôle : Figuration
Walk-on actors	Acteur : Figuration
Work	Œuvre

2) The code of the classes

```
#ifndef TypeCPP
#define TypeCPP

//---------------------------------------------------------------------------
#include <vcl.h>
#include <stdlib.h>
#include <except.h>
#include <math.h>
#include <time.h>
#pragma hdrstop

#include "Type.h"

//---------------------------------------------------------------------------

#pragma package(smart_init)
#pragma resource "*.dfm"
TWinType *WinType;
//---------------------------------------------------------------------------
__fastcall TWinType::TWinType(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

// type for all operations in Virtual Assistant
```

```
enum TVATools { // World
                E_tsNoTool,
                E_tsNewWorld, E_tsOpenWorld, E_tsSave, E_tsSaveAs,
                E_tsExport, E_tsQuit,

                //Stage
                E_tsFrontView, E_tsTopView, E_tsStageInfo,
                E_tsPlay, E_StageShape, E_tsObjectFloor, E_tsFloorObject,

                // Draw an Object
                E_tsCircle, E_tsFigure, E_tsImport, E_tsLine, E_tsFreehand, E_tsRectangle,

                // Modify an Object
                E_tsUndo, E_tsDeleteObject, E_tsDuplicate, E_tsPaste,
                E_tsGroup, E_tsFaceFront, E_tsFaceSide,
                E_tsMovePlan, E_tsMoveDeep, E_tsFlipHoriz, E_tsFlipVert,
                E_tsResize, E_tsCleanPixel, E_tsFixPattern, E_tsPattern,
                E_tsResizePattern, E_tsEraser, E_tsMaximise, E_tsTransparency,

                // Colour
                E_tsColourPick, E_tsColourCopy, E_tsColourWhite,
                E_tsColourBlack, E_tsFlood, E_tsTintObject, E_tsTintScene,
                E_tsLightObject, E_tsDarkObject, E_tsLightScene,
                E_tsDarkScene, E_tsEditFilter, E_tsMoveFilter,

                // Preferences
                E_tsPencilSize, E_tsOptions, E_tsBrushStyle

                // Help
              };
```

```
  // The Wireframe is SHOW or HIDE
  enum TWireframeMode { E_ShowWireFrame, E_HideWireFrame };

  // Top View or Front View
  enum TView { E_FrontView, E_TopView };

  // For the return of the functions
  enum TReturn { E_OK, E_ThereIsAProblem };

  // Display mode for the objects
  enum TDisplayMode { E_dmWhole, E_dmObjectClipRect_FrontView, E_dmObjectClipRect_TopView,
E_dmMergedClipRect };

  // Sense of the deep for the X-Axis
  enum TSenseX { E_BackLeftToFrontRigth, E_FrontLeftToBackRigth };

  // Sense of the deep for the Y-Axis
  enum TSenseY { E_FrontBottomToBackUp, E_FrontUpToBackDown };

  //---------------------------------------------------------------------------

  class TCoord_3D
  { // use the constructor, destructor and copy constructor by default
    // because you have no dynamic variable (with new and delete!)
    public : int X, Y, Z;
  };

  class TCoord_2D
  { // use the constructor, destructor and copy constructor by default
    // because you have no dynamic variable (with new and delete!)
    public : int X,Y;



  };

  class TMyDate
  { // use the constructor, destructor and copy constructor by default
    // because you have no dynamic variable (with new and delete!)
    public : int Month, Day, Year ;
  };

  //---------------------------------------------------------------------------

  // The objects on the Stage
  // Use this Class just in the Stage's Class
  class TStageObject
  { public : // NOT use the constructor, destructor and copy constructor by default
             // because you have some dynamic variables (with new and delete!)
             // I have not define the operator =, because I assign the pointer !!!
             TStageObject();  // Constructor
             TStageObject(const TStageObject & ObjectToCopy); // Copy Constructor
             ~TStageObject(); // Destructor

             void Set_NewPositionTop(TRect rectangle);
             void Set_NewPositionFront(TRect rectangle);
             TRect Get_NewPositionTop(void)        { return NewPositionTop; }
             TRect Get_NewPositionFront(void)      { return NewPositionFront; }
             TRect Get_PreviousPositionTop(void)   { return PreviousPositionTop; }
             TRect Get_PreviousPositionFront(void) { return PreviousPositionFront; }
             TRect Get_MergedRectangleFront(void)  { return MergedRectangleFront; }
             TRect Get_MergedRectangleTop(void)    { return MergedRectangleTop; }

             TGraphic* Get_StorageOriginalBMP(void) { return StorageOriginalBMP; }
```

```
            void Set_StorageOriginalBMP(AnsiString FileName) { StorageOriginalBMP->LoadFromFile( File-
Name ); }
            TCanvas* Get_StorageOriginalBMPCanvas(void) { return StorageOriginalBMP->Canvas; }

            void Set_StorageOriginalBMPWidth(int Width) { StorageOriginalBMP->Width = Width; }
            int  Get_StorageOriginalBMPWidth(void) { return StorageOriginalBMP->Width; }
            void Set_StorageOriginalBMPHeight(int Height) { StorageOriginalBMP->Height = Height; }
            int  Get_StorageOriginalBMPHeight(void) { return StorageOriginalBMP->Height; }
            bool Get_StorageOriginalBMPTransparency(void) { return StorageOriginalBMP->Transparent; }

            void Set_Size_3D(int X, int Y, int Z) { Size_3D.X = X; Size_3D.Y = Y; Size_3D.Z = Z; }
            int Set_Size_3D_X(int X) { Size_3D.X = X; }
            int Set_Size_3D_Y(int Y) { Size_3D.Y = Y; }
            int Set_Size_3D_Z(int Z) { Size_3D.Z = Z; }
            int Get_Size_3D_X(void) { return Size_3D.X; }
            int Get_Size_3D_Y(void) { return Size_3D.Y; }
            int Get_Size_3D_Z(void) { return Size_3D.Z; }

            void Set_Center_3D(int X, int Y, int Z) { Center_3D.X = X; Center_3D.Y = Y; Center_3D.Z =
Z; }
            int Set_Center_3D_X(int X) { Center_3D.X = X; }
            int Set_Center_3D_Y(int Y) { Center_3D.Y = Y; }
            int Set_Center_3D_Z(int Z) { Center_3D.Z = Z; }
            TCoord_3D Get_Center_3D(void) { return Center_3D; }
            int Get_Center_3D_X(void) { return Center_3D.X; }
            int Get_Center_3D_Y(void) { return Center_3D.Y; }
            int Get_Center_3D_Z(void) { return Center_3D.Z; }

            int Get_IDentifier(void) { return IDentifier; }

            void Set_StorageOriginalBMPCopyRect(TCanvas* Canvas, TRect Source);
```

```
  //          void RePosition_NewPositionTopAndPreviousPositionTop(int argument);
            void ReleaseTextOfTheStageObjectForSaving(void);

            // Number of StageObject for all Stage !!!
            static int Get_NumberOfStageObjectInstantiedInTheClass(void) { return TSta-
geObect::NumberOfStageObjectInstantiedInTheClass; }

    private : int IDentifier; // Identifier of the StageObject in the list
  //            int TimeStart;
  //            int TimeEndStop;
             Graphics::TBitmap* StorageOriginalBMP; // Picture of the StageObject on the screen
  //            Graphics::TBitmap* StoragePatternedBMP;
  //            Graphics::TBitmap* StorageAngledBMP;
             bool Patterned;
             TCoord_3D Center_3D;
             TCoord_3D Size_3D;
             int OffSetHoriz;
             int OffSetVert;
             bool BillBoard;
             TRect PreviousPositionFront;
             TRect PreviousPositionTop;
             TRect NewPositionFront;
             TRect NewPositionTop;
             TRect MergedRectangleFront;
             TRect MergedRectangleTop;

  //            TRect ImageTop;

             void Set_MergedRectangleFront(void);
             void Set_MergedRectangleTop(void);
```

```
            void Set_IDentifier(int NewIDentifier) { IDentifier = NewIDentifier; }

            // GLOBAL VARIABLES -- CLASS' VARIABLES
            // THIS IS THE -- DECLARATION --
            // How many instancieted StageObject do you have?
            // Number of StageObject for all Stage !!!
            static int NumberOfStageObjectInstantiedInTheClass;
            static void IncreaseOfOne_NumberOfStageObjectInstantiedInTheClass(void) { TStageOb-
ject::NumberOfStageObjectInstantiedInTheClass++; }
            static TReturn DecreaseOfOne_NumberOfStageObjectInstantiedInTheClass(void);

            // This class is a friend because she changes the IDentifer
            friend class TStage;
  };

  TStageObject::TStageObject()
  { IncreaseOfOne_NumberOfStageObjectInstantiedInTheClass();
    Set_IDentifier(NumberOfStageObjectInstantiedInTheClass - 1); // the first IDentifier is 0 !

    // initialisation of variables

    // initialization of rectangles
    PreviousPositionFront = Rect(0,0,0,0);
    PreviousPositionTop = Rect(0,0,0,0);
    NewPositionFront = Rect(0,0,0,0);
    NewPositionTop = Rect(0,0,0,0);
  //  ImageTop = Rect(0,0,0,0);

    // BitMaps' Allocation
    StorageOriginalBMP = new Graphics::TBitmap;
    StorageOriginalBMP->TransparentColor = clWhite;



    StorageOriginalBMP->Transparent = true;
    StorageOriginalBMP->Canvas->CopyMode = cmSrcCopy;

    Set_Size_3D(StorageOriginalBMP->Height, StorageOriginalBMP->Width, 0);
    Set_Center_3D(NewPositionFront.Left + (StorageOriginalBMP->Width / 2), NewPositionFront.Top +
(StorageOriginalBMP->Height / 2), 0);
  }

  TStageObject::~TStageObject()
  { TStageObject::DecreaseOfOne_NumberOfStageObjectInstantiedInTheClass();

    // Delete the Bitmap
    delete StorageOriginalBMP;

  }

  TStageObject::TStageObject(const TStageObject & ObjectToCopy) // Copy Constructor
  { // Just the IDentificator is different between the both!!!
    IncreaseOfOne_NumberOfStageObjectInstantiedInTheClass();
    Set_IDentifier( NumberOfStageObjectInstantiedInTheClass - 1 ); // the first IDentifier is 0 !

    //            int TimeStart;
    //            int TimeEndStop;

    // I have seen no copy constructor in the bitmap's class !!!
    StorageOriginalBMP = new Graphics::TBitmap;
    StorageOriginalBMP->Assign(ObjectToCopy.StorageOriginalBMP);

    //            Graphics::TBitmap* StoragePatternedBMP;
    //            Graphics::TBitmap* StorageAngledBMP;
```

```
    Patterned = ObjectToCopy.Patterned;

    // Use the copy constructor of TCoord_3D !!!
    Center_3D = ObjectToCopy.Center_3D;
    Size_3D = ObjectToCopy.Size_3D;

    OffSetHoriz = ObjectToCopy.OffSetHoriz;
    OffSetVert = ObjectToCopy.OffSetVert;
    BillBoard = ObjectToCopy.BillBoard;

    // Use the copy constructor of TRect !!!
    PreviousPositionFront = ObjectToCopy.PreviousPositionFront;
    PreviousPositionTop = ObjectToCopy.PreviousPositionTop;
    NewPositionFront = ObjectToCopy.NewPositionFront;
    NewPositionTop = ObjectToCopy.NewPositionTop;
    //            TRect ImageTop;
  }

  void TStageObject::Set_NewPositionTop(TRect rectangle)
  { PreviousPositionTop = NewPositionTop;
    NewPositionTop = rectangle;
    Set_MergedRectangleTop();
  }

  void TStageObject::Set_NewPositionFront(TRect rectangle)
  { PreviousPositionFront = NewPositionFront;
    NewPositionFront = rectangle;
    Set_MergedRectangleFront();
  }

  void TStageObject::Set_MergedRectangleFront(void)

  { MergedRectangleFront.Left   = min(PreviousPositionFront.Left,   NewPositionFront.Left);
    MergedRectangleFront.Top    = min(PreviousPositionFront.Top,    NewPositionFront.Top);
    MergedRectangleFront.Right  = max(PreviousPositionFront.Right,  NewPositionFront.Right);
    MergedRectangleFront.Bottom = max(PreviousPositionFront.Bottom, NewPositionFront.Bottom);
  }

  void TStageObject::Set_MergedRectangleTop(void)
  { MergedRectangleTop.Left     = min(PreviousPositionTop.Left,   NewPositionTop.Left);
    MergedRectangleTop.Top      = min(PreviousPositionTop.Top,    NewPositionTop.Top);
    MergedRectangleTop.Right    = max(PreviousPositionTop.Right,  NewPositionTop.Right);
    MergedRectangleTop.Bottom   = max(PreviousPositionTop.Bottom, NewPositionTop.Bottom);
  }

  void TStageObject::Set_StorageOriginalBMPCopyRect(TCanvas* Canvas, TRect Source)
  { // Specifies how the a graphical image is copied onto the canvas
    // cmSrcCopy = Copies the source bitmap to the canvas
    StorageOriginalBMP->Canvas->CopyMode = cmSrcCopy;
    StorageOriginalBMP->Canvas->CopyRect(Rect(0, 0, StorageOriginalBMP->Width, StorageOriginalBMP-
>Height), Canvas, Source);
  }

  /*void TStageObject::RePosition_NewPositionTopAndPreviousPositionTop(int argument)
  { int HalfWidth = Get_StorageOriginalBMPWidth() / 2;
    NewPositionTop = PreviousPositionTop = Rect( Center_3D.X - HalfWidth, (argument - Center_3D.Z) - 6,
Center_3D.X + HalfWidth, (argument - Center_3D.Z) + 6 );
  } */

  TReturn TStageObject::DecreaseOfOne_NumberOfStageObjectInstantiedInTheClass(void)
  { if (TStageObject::NumberOfStageObjectInstantiedInTheClass >= 1)
      { TStageObject::NumberOfStageObjectInstantiedInTheClass--;
```

(5)

```
      return E_OK;
    }
  else return E_ThereIsAProblem;
}

void TStageObject::ReleaseTextOfTheStageObjectForSaving(void)
{  char IntegerToAnString[15];
   char AnString[1024];

  // I use an object " memo " to stock the text that should be saved.
  // Why? Because it is comfortable to manipulate a text in an "memo"
  // and an "memo" have no limit with the size of the text ...
  // Besides, this object has its visible " property " to false.

  WinType->TextStageObjectSaving->Lines->SetText(""); // the list of strings is Empty

  // Flag for the Beginning of the StageObject
  AnString[0] = '\0'; // The String is empty
  StrCat(AnString, "[StageObject]");

  //StrCat(AnString, "{TimeStart}");

  //StrCat(AnString, "{TimeEndStop}");
//++
//  StrCat(AnString, "{StorageOriginalBMP}");
//  StorageOriginalBMP->SaveToFile(StrCat("", itoa(IDentifier)))
//++
  //StrCat(AnString, "{StoragePatternedBMP}");

  //StrCat(AnString, "{StorageAngledBMP}");
```

```
  StrCat(AnString, "{Patterned}");
  if (Patterned == true)
   { StrCat(AnString, "True");
   }
  else
   { StrCat(AnString, "False");
   }

  StrCat(AnString, "{Center_3D}");

  itoa(Center_3D.X, IntegerToAnString, 10);
  StrCat(AnString, IntegerToAnString);
  StrCat(AnString, ";");
  itoa(Center_3D.Y, IntegerToAnString, 10);
  StrCat(AnString, IntegerToAnString);
  StrCat(AnString, ";");
  itoa(Center_3D.Z, IntegerToAnString, 10);
  StrCat(AnString, IntegerToAnString);

  StrCat(AnString, "{Size_3D}");
  itoa(Size_3D.X, IntegerToAnString, 10);
  StrCat(AnString, IntegerToAnString);
  StrCat(AnString, ";");
  itoa(Size_3D.Y, IntegerToAnString, 10);
  StrCat(AnString, IntegerToAnString);
  StrCat(AnString, ";");
  itoa(Size_3D.Z, IntegerToAnString, 10);
  StrCat(AnString, IntegerToAnString);

(6)
  StrCat(AnString, "{OffSetHoriz}");
```

```
itoa(OffSetHoriz, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);

StrCat(AnString, "{OffSetVert}");
itoa(OffSetVert, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);

StrCat(AnString, "{BillBoard}");
if (BillBoard) StrCat(AnString, "True");
else  StrCat(AnString, "False");

StrCat(AnString, "{PreviousPositionFront}");
itoa(PreviousPositionFront.Left, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);
StrCat(AnString, ";");
itoa(PreviousPositionFront.Top, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);
StrCat(AnString, ";");
itoa(PreviousPositionFront.Right, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);
StrCat(AnString, ";");
itoa(PreviousPositionFront.Bottom, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);

StrCat(AnString, "{PreviousPositionTop}");
itoa(PreviousPositionTop.Left, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);
StrCat(AnString, ";");
itoa(PreviousPositionTop.Top, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);
StrCat(AnString, ";");
```

```
itoa(PreviousPositionTop.Right, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);
StrCat(AnString, ";");
itoa(PreviousPositionTop.Bottom, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);

StrCat(AnString, "{NewPositionFront}");
itoa(NewPositionFront.Left, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);
StrCat(AnString, ";");
itoa(NewPositionFront.Top, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);
StrCat(AnString, ";");
itoa(NewPositionFront.Right, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);
StrCat(AnString, ";");
itoa(NewPositionFront.Bottom, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);

StrCat(AnString, "{NewPositionTop}");
itoa(NewPositionTop.Left, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);
StrCat(AnString, ";");
itoa(NewPositionTop.Top, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);
StrCat(AnString, ";");
itoa(NewPositionTop.Right, IntegerToAnString, 10);
StrCat(AnString, IntegerToAnString);
StrCat(AnString, ";");
itoa(NewPositionTop.Bottom, IntegerToAnString, 10);
```

```
    StrCat(AnString, IntegerToAnString);

    StrCat(AnString, "{MergedRectangleFront}");
    itoa(MergedRectangleFront.Left, IntegerToAnString, 10);
    StrCat(AnString, IntegerToAnString);
    StrCat(AnString, ";");
    itoa(MergedRectangleFront.Top, IntegerToAnString, 10);
    StrCat(AnString, IntegerToAnString);
    StrCat(AnString, ";");
    itoa(MergedRectangleFront.Right, IntegerToAnString, 10);
    StrCat(AnString, IntegerToAnString);
    StrCat(AnString, ";");
    itoa(MergedRectangleFront.Bottom, IntegerToAnString, 10);
    StrCat(AnString, IntegerToAnString);

    StrCat(AnString, "{MergedRectangleTop}");
    itoa(MergedRectangleTop.Left, IntegerToAnString, 10);
    StrCat(AnString, IntegerToAnString);
    StrCat(AnString, ";");
    itoa(MergedRectangleTop.Top, IntegerToAnString, 10);
    StrCat(AnString, IntegerToAnString);
    StrCat(AnString, ";");
    itoa(MergedRectangleTop.Right, IntegerToAnString, 10);
    StrCat(AnString, IntegerToAnString);
    StrCat(AnString, ";");
    itoa(MergedRectangleTop.Bottom, IntegerToAnString, 10);
    StrCat(AnString, IntegerToAnString);

    // Add the new String at the end of the list
    // Append() is the same of Add(), but with no return value
    WinType->TextStageObjectSaving->Lines->SetText(AnString);
```

```
  }

  // variables' Allocation
  // THIS IS THE -- DEFINITION --
  int TStageObject::NumberOfStageObjectInstantiedInTheClass = 0; // When it's initialise to 0, the com-
piler initialise the variable to 1 !?!

  //---------------------------------------------------------------------------

  // Class for slide of the Stage
  class TStage
  { public : TStage();
              ~TStage();
              TStage(const TStage & StageToCopy);

             TStageObject * Add_NewStageObject(void); // return the New StageObject created
             TStageObject * Duplicate_StageObject(TStageObject * ObjectToCopy); // Create a NewStageOb-
ject and Duplicate it
             void Delete_StageObjectNumber(int IDentifier); // Free the StageObject and Put it out of
the List
             void Delete_StageObjectNumberInTheList(int IDentifier); // Put an Object out of the List
             void Insertion(int Position,  TStageObject * StageObjectToInsert);
             void UpgradeTheIDentifierOfAllObjectSince(int SinceObject);
             void ReleaseTextOfTheStageForSaving(void);

  //           void ShiftStageObject(int PreviousPosition, int NewPosition);
             TStageObject * Get_StageObjectNumber(int IDentifier);
             TStageObject * Get_LastStageObject(void);
             int Get_NumberOfStageObjectInThisStage(void);
```

```
            TStageObject * Get_SeletedObjectByTheMouse(void) { return SeletedObjectByTheMouse; }
            void Set_SeletedObjectByTheMouse(TStageObject * StageObject) { SeletedObjectByTheMouse =
StageObject; }

            TColor Get_StageColour(void)            { return StageColour; }
            void   Set_StageColour(TColor Argument) { StageColour = Argument; }

            TColor Get_FloorColour(void)            { return FloorColour; }
            void   Set_FloorColour(TColor Argument) { FloorColour = Argument; }

    private : TList * ListOfStageObjectInThisStage;
              TStageObject * SeletedObjectByTheMouse; // Used by the three event on the Stage : Mouse-
Down, MouseMove, MouseUp
              TColor FloorColour; // TColor = Signed Integer on 4 bytes ...
              TColor StageColour;
  };

  TStage::TStage() // Default Constructor
  { ListOfStageObjectInThisStage = new TList;
    SeletedObjectByTheMouse = NULL;
  }

  TStage::~TStage() // Destructor
  { ListOfStageObjectInThisStage->Clear();
    // I don't know if clear() do the delete or not !?!
    delete ListOfStageObjectInThisStage;
  }

  TStage::TStage(const TStage & StageToCopy) // Copy Constructor
  { int Sweeper;
    TStageObject * ObjectToCopy;
```

```
    // Copy all StageObjects
    for (Sweeper = 0; Sweeper < StageToCopy.Get_NumberOfStageObjectInThisStage(); Sweeper++)
     { ObjectToCopy = StageToCopy.Get_StageObjectNumber(Sweeper);
       // create a new StageObject "CopiedObject" by copying "ObjectToCopy"
       TStageObject * CopiedObject(ObjectToCopy);
       ListOfStageObjectInThisStage->Add((void *) CopiedObject);
     }

    SeletedObjectByTheMouse = StageToCopy.SeletedObjectByTheMouse;
    FloorColour =  StageToCopy.FloorColour;
    StageColour = StageToCopy.StageColour;
  }

  TStageObject * TStage::Add_NewStageObject(void)
  { TStageObject * NewStageObject;
    NewStageObject = new TStageObject;

    // The index begin at 0 !!!
    ListOfStageObjectInThisStage->Add((void *) NewStageObject);
    // Put the New as Current : copy the pointer, not the StageObject
    SeletedObjectByTheMouse = NewStageObject;

    return NewStageObject;
  }

  TStageObject * TStage::Duplicate_StageObject(TStageObject *  ObjectToCopy)
  { // Copy Constructor used
    TStageObject * NewStageObject;
    NewStageObject = new TStageObject(* ObjectToCopy);
```

```
    // The index begin at 0 !!!
    ListOfStageObjectInThisStage->Add((void *) NewStageObject);

    // Put the New as Current : copy the pointer, not the StageObject
    SeletedObjectByTheMouse = NewStageObject;

    return NewStageObject;
  }

  void TStage::Delete_StageObjectNumber(int IDentifier)
  { TStageObject * StageObjectToDelete, * StageObjectAtTheSweeper;
    int Sweeper;

    // Delete the specified StageObject and change the Count and
    // IDentifier of the other StageObjects
    StageObjectToDelete = Get_StageObjectNumber(IDentifier);
    delete StageObjectToDelete;
    // Delete the pointer in the list and the list change all IDentifier after
    // the DeletedObject : 0 1 2 3 4 5 6          count = 7
    //             Delete : 0 1 x 3 4 5 6          count = 6
    //             After :  0 1   2 3 4 5          count = 6
    ListOfStageObjectInThisStage->Delete(IDentifier);
    // Change all IDentifier property of the StageObject after the DeletedObject
    // because the TList have changed the IDentifier property in TList
    for (Sweeper = IDentifier; Sweeper < Get_NumberOfStageObjectInThisStage(); Sweeper++)
     { StageObjectAtTheSweeper = Get_StageObjectNumber(Sweeper);
       // TStage is A FRIEND CLASS !!!!
       StageObjectAtTheSweeper->Set_IDentifier( Sweeper );
     }
  }
```

```
  void TStage::Delete_StageObjectNumberInTheList(int IDentifier) // Put an Object out of the List and
NOT delete it
  { // Delete the pointer in the list and the list change all IDentifier after of the other objects
    // the DeletedObject : 0 1 2 3 4 5 6          count = 7
    //             Delete : 0 1 x 3 4 5 6          count = 6
    //             After :  0 1   2 3 4 5          count = 6
    ListOfStageObjectInThisStage->Delete(IDentifier);

    // Change all IDentifier property of the StageObject after the DeletedObject
    // because the TList have changed the IDentifier property in TList
    UpgradeTheIDentifierOfAllObjectSince(IDentifier);
  }

  void TStage::Insertion(int Position, TStageObject * StageObjectToInsert)
  { if (Position == Get_NumberOfStageObjectInThisStage())
      { ListOfStageObjectInThisStage->Add((void *) StageObjectToInsert);
        StageObjectToInsert->Set_IDentifier(Get_NumberOfStageObjectInThisStage() - 1);  // This class
is a Friend
      }
    else
      { ListOfStageObjectInThisStage->Insert(Position, (void *) StageObjectToInsert);
        StageObjectToInsert->Set_IDentifier(Position); // This class is a Friend
        // Change all IDentifier property of the StageObject after the InsertedObject
        // because the TList have changed the IDentifier property in TList
        UpgradeTheIDentifierOfAllObjectSince(Position + 1);
      }
  }

  void TStage::UpgradeTheIDentifierOfAllObjectSince(int SinceObject)
```

```
{ int Sweeper;
  TStageObject * StageObjectAtTheSweeper;

  // Change all IDentifier property of the StageObject after the Deleted, Inserted, ..., Object
  // because the TList have changed the IDentifier property in TList
  for (Sweeper = SinceObject; Sweeper < Get_NumberOfStageObjectInThisStage(); Sweeper++)
   { StageObjectAtTheSweeper = Get_StageObjectNumber(Sweeper);
     // TStage is A FRIEND CLASS !!!!
     StageObjectAtTheSweeper->IDentifier = Sweeper;
   }
}

void TStage::ReleaseTextOfTheStageForSaving(void)
{ // I use an object " memo " to stock the text that should be saved.
  // Why? Because it is comfortable to manipulate lines of strings
  // with this type of object.
  // Besides, this object has its visible " property " to false.

  int Sweeper;
  TStageObject * StageObjectAtCurrentSweeper;
  char AnString[1024];
  char IntegerToChar[15];

  WinType->TextStageSaving->Lines->SetText(""); // all the memo is Empty
  AnString[0] = '\0'; // The string is now empty

  // Flag for the Beginning of the Stage
  // the first line of the TextforSaving is reserved
  // for the properties of the Stage
  StrCat( AnString, "[Stage]" );
```

```
  StrCat( AnString, "{FloorColour}");
  itoa(FloorColour, IntegerToChar, 10);
  StrCat( AnString, IntegerToChar);

  StrCat( AnString, "{StageColour}");
  itoa(StageColour, IntegerToChar, 10);
  StrCat( AnString, IntegerToChar);

  WinType->TextStageSaving->Lines->SetText(AnString);

  for (Sweeper = 0; Sweeper < Get_NumberOfStageObjectInThisStage(); Sweeper++)
   { // Take the StageObject at the Current Sweeper
     StageObjectAtCurrentSweeper = Get_StageObjectNumber(Sweeper);
     StageObjectAtCurrentSweeper->ReleaseTextOfTheStageObjectForSaving();
     // After the ReleaseText... function, the property Text of
     // the meme WinType->TextStageObjectSaving contain the Text fo the current StageObject

     // Add the New ReleaseText to the end of the text of the memo
     WinType->TextStageSaving->Lines->Append(WinType->TextStageObjectSaving->Lines->GetText());
   }

}

/*// THIS FUNCTION DOESN'T RUN : YOU HAVE NEED OF TO CHANGE ALSO THE IDENTIFIER
void TStage::ShiftStageObject(int PreviousPosition, int NewPosition)
{ // Changes the position of two items in the list of StageObjects
  ListOfStageObjectInThisStage->Exchange(PreviousPosition, NewPosition);
} */

TStageObject * TStage::Get_StageObjectNumber(int IDentifier)
```

```
{ // Items from 0 to [number of StageObjects - 1]
  if ( IDentifier < Get_NumberOfStageObjectInThisStage() )
    return (TStageObject *) ListOfStageObjectInThisStage->Items[IDentifier];
  else
    return NULL;
}

TStageObject * TStage::Get_LastStageObject(void)
{ int Number;

  Number = Get_NumberOfStageObjectInThisStage();

  // You have at the minimum one object !
  if ( Number > 0 )
    return (TStageObject *) ListOfStageObjectInThisStage->Items[Number - 1];
  else
    return NULL;
}

int TStage::Get_NumberOfStageObjectInThisStage(void)
{ return ListOfStageObjectInThisStage->Count;
}

//---------------------------------------------------------------------------

// Class for the World
class TWorld
{ public : TWorld();
           ~TWorld();

           int  Add_NewStage(void); // return the IDentifier of the New Slide



           void Delete_Stage(int IDentifier);

           int  Get_NumberOfStageOnThisWorld(void);
           TStage * Get_StageNumber(int IDentifier);
           TStage * Get_CurrentStage(void) { return CurrentStage; }

           void Set_CurrentStage(TStage * Stage) { CurrentStage = Stage; }
                                       // Copy just the 31 first caracters !!!
           void Set_WorldName(char * Name) { strncpy(WorldName, Name, 31); }
                                      // Copy just the 31 first caracters !!!
           void Set_Author(char * Name) { strncpy(Author, Name, 31); }
           void Set_DateCreation(TMyDate Datum) { DateCreation.Year  = Datum.Year;
                                                  DateCreation.Month = Datum.Month;
                                                  DateCreation.Day   = Datum.Day; }
           void Set_DateCreationYear(int Year) { DateCreation.Year  = Year; }
           void Set_DateCreationMonth(int Month) { DateCreation.Month  = Month; }
           void Set_DateCreationDay(int Day) { DateCreation.Day  = Day; }

           void ReleaseTextOfTheWorldForSaving(void); // Create the Text of the world for saving in
the memo TexwtorldSaving

   private : TList * ListOfStageOnThisWorld;
             TStage * CurrentStage;
             char WorldName[35];
             char Author[35];
             TMyDate DateCreation;
};

TWorld::TWorld()
{ ListOfStageOnThisWorld = new TList;
```

```
  CurrentStage = NULL;

  // IT is possible for to create a constructor with parameters
  // if you use the new keyword for an dynamical use ???
  strcpy(Author,"");
  strcpy(WorldName,"");
  DateCreation.Day = 0;DateCreation.Month = 0;DateCreation.Year = 0;
}

TWorld::~TWorld()
{ ListOfStageOnThisWorld->Clear();
  delete ListOfStageOnThisWorld;
}

int TWorld::Add_NewStage(void)
{ TStage * NewStage;
  NewStage = new TStage;

  // The New Stage is now the Current : copy the pointer, not the stage !
  CurrentStage = NewStage;
  // The index begin at 0 !!!
  return ListOfStageOnThisWorld->Add((void *) NewStage);
}

void TWorld::Delete_Stage(int IDentifier)
{ // Delete the specified StageObject and change the Count and
  // IDentificater of the other StageObjects
  ListOfStageOnThisWorld->Delete(IDentifier);
}

int TWorld::Get_NumberOfStageOnThisWorld(void)
{ return ListOfStageOnThisWorld->Count;
}

TStage * TWorld::Get_StageNumber(int IDentifier)
{ // Items from 0 to [number of Stages - 1]
  if ( IDentifier < Get_NumberOfStageOnThisWorld() )
    return (TStage *) ListOfStageOnThisWorld->Items[IDentifier];
  else
    return NULL;
}

void PutCurrentDate(TMyDate * DateCreation)
{ time_t Now = time(0); // Get current Date and Time
  struct tm * DateAndTime = localtime(&Now); // Convertion

  DateCreation->Month = DateAndTime->tm_mon + 1;
  DateCreation->Day   = DateAndTime->tm_mday;
  DateCreation->Year  = 1900 + DateAndTime->tm_year;
}

void TWorld::ReleaseTextOfTheWorldForSaving(void)
{ TStage * StageAtTheCurrentSweeper;
  int Sweeper;
  char AnString[1024];
  char IntegerToChar[10];

  WinType->TextStageSaving->Lines->SetText(""); // all the memo is Empty
  AnString[0] = '\0'; // The string is now empty

  // Flag for the copyrigth of Visual Assistant
```

```
    StrCat( AnString, "[Visual Assistant 1.0]{MacVersion}Colin BEAR-
DON{PCVersion}Vincent_FONTAINE@yahoo.com" );
    WinType->TextWorldSaving->Lines->SetText(AnString);

    AnString[0] = '\0'; // The string is now empty
    // Flag for the Beginning of the World
    StrCat( AnString, "[World]" );
    StrCat( AnString, "{WorldName}");
    StrCat( AnString, WorldName);
    StrCat( AnString, "{Author}");
    StrCat( AnString, Author);

    StrCat( AnString, "{DateCreation}Month:");
    // if there are no saving before, I put the current date
    if ((DateCreation.Month == 0) && (DateCreation.Day == 0) && (DateCreation.Year == 0)) PutCurrent-
Date(&DateCreation);

    itoa(DateCreation.Month, IntegerToChar, 10);
    StrCat( AnString, IntegerToChar);
    StrCat( AnString, "/Day:");
    itoa(DateCreation.Day, IntegerToChar, 10);
    StrCat( AnString, IntegerToChar);
    StrCat( AnString, "/Year:");
    itoa(DateCreation.Year, IntegerToChar, 10);
    StrCat( AnString, IntegerToChar);

    // Put the properties of the World in the first line
    WinType->TextWorldSaving->Lines->Append(AnString);

    // Items from 0 to [number of Stages - 1]
    for (Sweeper = 0; Sweeper < Get_NumberOfStageOnThisWorld(); Sweeper++)
     { StageAtTheCurrentSweeper = Get_StageNumber(Sweeper);
       // Make the Text for the Stage at the current Sweeper in the memo TextStageSaving
       StageAtTheCurrentSweeper->ReleaseTextOfTheStageForSaving();
       // Call AddStrings to add the strings from another TStrings object to the list
       WinType->TextWorldSaving->Lines->Append(WinType->TextStageSaving->Lines->GetText());
     }

    // Put the EOF at the beginning of a new line
    // PS : When you append nothing, you add an hide '\n' ...
    WinType->TextWorldSaving->Lines->Append("");
  }

  #endif
```

3) The Lexer code

```
/* Scanner for the definition of the Visual Assistant Saving Files */

/* Composition of this file :           */
/*      Definition Section              */
/*      %%                              */
/*      Rules Section = Pattern Action  */
/*      %%                              */
/*      User Subroutines                */
/*                                      */

/* The Parser copy the " %{ %} "  completely to the output C file */
%{ #include <stdio.h>
   #include <string.h>
   #include "lexer.h";

   #define HEADER    70575
   #define PROPERTY  28178
   #define ENDOFFILE 66668

   /* if you want to debug the file, define DEBUGGING_LEXER and the FILE */
   #define DEBUGGING_LEXER
   FILE * FICHIER;

   /* You have also the debug of FLEX. For to use it, define FLEX_DEBUG */
   /* #define FLEX_DEBUG */

   /* void yyerror(char *s); */
   char endroit[100];
%}

/* Definition of all regular start states */
%x VA_STATE VA_STATE_1 VA_STATE_2 VA_STATE_3 EXITVA_STATE
```

```
%x WORLD_STATE WORLDSTRING_STATE WORLDDATE_STATE WORLDDATEVALUE_STATE EXITWORLD_STATE
%x STAGE_STATE STAGEVALUE_STATE EXITSTAGE_STATE
%x STAGEOBJECT_STATE STAGEOBJECT_INTEGER_VALUE_STATE STAGEOBJECT_3D_COORDONATE_VALUE_STATE
%x STAGEOBJECT_BOOLEAN_VALUE_STATE STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE
%x ENDOFLEXER_STATE


%%

          /* ************************************************************************ */
          /* ****************** Visual Assistant Line ******************************* */
          /* ************************************************************************ */

          /* BEGIN is the "INITIAL" State                                  */
          /* ^ denotes the pattern must start at the beginning of a line */
          /* Doesn't run with the STATES ?!?                               */
^"[Visual Assistant 1.0]"               { BEGIN VA_STATE;
                                          strcpy(endroit, "Header:[VA]");
                                          #if defined(DEBUGGING_LEXER)
                                            fprintf(FICHIER,"#HEADER : %s#\n",endroit);
                                          #endif
                                        }

<VA_STATE>"{MacVersion}"                { BEGIN VA_STATE_1;
                                          strcpy(endroit, "Property:{MacVersion}");
                                          #if defined(DEBUGGING_LEXER)
                                            fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                          #endif
                                        }

<VA_STATE_1>"Colin BEARDON"             { BEGIN VA_STATE_2;
                                          strcpy(endroit, "MacVersionColin");
                                          #if defined(DEBUGGING_LEXER)
                                            fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                          #endif
                                        }
```


Appendix - the lexer code

```
<VA_STATE_2>"{PCVersion}"                { BEGIN VA_STATE_3;
                                           strcpy(endroit, "Property:{PCVersion}");
                                           #if defined(DEBUGGING_LEXER)
                                             fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                           #endif
                                         }

<VA_STATE_3>"Vincent_FONTAINE@yahoo.com" { BEGIN EXITVA_STATE;
                                           strcpy(endroit, "PCVersionVincent");
                                           #if defined(DEBUGGING_LEXER)
                                             fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                           #endif
                                         }


        /* ************************************************************************ */
        /* ***************************** World Line ******************************* */
        /* ************************************************************************ */


<EXITVA_STATE>"[World]{"           { BEGIN WORLD_STATE;
                                     strcpy(endroit, "Header:[World]");
                                     #if defined(DEBUGGING_LEXER)
                                       fprintf(FICHIER,"#HEADER : %s#\n",endroit);
                                     #endif
                                   }

<WORLD_STATE>"WorldName}"          { BEGIN WORLDSTRING_STATE;
                                     strcpy(endroit, "Property:{WorldName}");
                                     #if defined(DEBUGGING_LEXER)
                                       fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                     #endif
                                   }

<WORLD_STATE>"Author}"             { BEGIN WORLDSTRING_STATE;
                                     strcpy(endroit, "Property:{Author}");
```

```
                                     #if defined(DEBUGGING_LEXER)
                                       fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                     #endif
                                   }

<WORLD_STATE>"DateCreation}"       { BEGIN WORLDDATE_STATE;
                                     strcpy(endroit, "Property:{DateCreation}");
                                     #if defined(DEBUGGING_LEXER)
                                       fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                     #endif
                                   }

        /* you needn't to place { between "" because you are in the [] */
        /* In this context ( between [] ), ^ denotes the negation !!! */
<WORLDSTRING_STATE>[^{\n]*"{"      { BEGIN WORLD_STATE;
                                     yytext[yyleng - 1] = '\0'; /* Put out the "{"         */
                                                                /* yyleng = strlen(yytext) */
                                     #if defined(DEBUGGING_LEXER)
                                       fprintf(FICHIER,"#WORLDSTRING { : %s#\n",yytext);
                                     #endif
                                   }

<WORLDSTRING_STATE>[^{\n]*\n       { BEGIN STAGE_STATE;
                                     yytext[yyleng - 1] = '\0'; /* Put out the "{"         */
                                                                /* yyleng = strlen(yytext) */
                                     #if defined(DEBUGGING_LEXER)
                                       fprintf(FICHIER,"#WORLDSTRING \\n : %s#\n",yytext);
                                     #endif
                                   }

        /* You may change the order of the properties for the date */
        /* MM/JJ/YY or JJ/MM/YY or ...                             */
<WORLDDATE_STATE>"Month:"          { BEGIN WORLDDATEVALUE_STATE; /* Go for read the integer for the Month */
                                     strcpy(endroit, "Property:{DateCreation}:Month");
                                     #if defined(DEBUGGING_LEXER)
                                       fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
```

```
                                #endif
                              }

<WORLDDATE_STATE>"Day:"       { BEGIN WORLDDATEVALUE_STATE; /* Go for read the integer for the Day */
                                strcpy(endroit, "Property:{DateCreation}:Day");
                                #if defined(DEBUGGING_LEXER)
                                  fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                #endif
                              }

<WORLDDATE_STATE>"Year:"      { BEGIN WORLDDATEVALUE_STATE; /* Go for read the integer for the Year */
                                strcpy(endroit, "Property:{DateCreation}:Year");
                                #if defined(DEBUGGING_LEXER)
                                  fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                #endif
                              }

<WORLDDATEVALUE_STATE>[0-9]*\n  { BEGIN EXITWORLD_STATE;     /* Exit the World property */
                                yytext[yyleng - 1] = '\0'; /* Put out the "\n"        */
                                #if defined(DEBUGGING_LEXER)
                                   fprintf(FICHIER,"#WORLDDATE \\n : %s#\n",yytext);
                                #endif
                              }

<WORLDDATEVALUE_STATE>[0-9]*"{"  { BEGIN WORLD_STATE;         /* Return to World property */
                                yytext[yyleng - 1] = '\0'; /* Put out the "{"         */
                                #if defined(DEBUGGING_LEXER)
                                   fprintf(FICHIER,"#WORLDDATE { : %s#\n",yytext);
                                #endif
                              }

<WORLDDATEVALUE_STATE>[0-9]*"/"  { BEGIN WORLDDATE_STATE;           /* Return to Date Property */
                                yytext[yyleng - 1] = '\0'; /* Put out the "/"         */
                                #if defined(DEBUGGING_LEXER)
                                   fprintf(FICHIER,"#WORLDDATE / : %s#\n",yytext);
                                #endif
                              }
```

```
        /* ************************************************************************ */
        /* ***************************** Stage Line ******************************* */
        /* ************************************************************************ */


<EXITWORLD_STATE>"[Stage]{"   { BEGIN STAGE_STATE;
                                strcpy(endroit, "Header:[Stage]");
                                #if defined(DEBUGGING_LEXER)
                                  fprintf(FICHIER,"#HEADER : %s#\n",endroit);
                                #endif
                              }

<STAGE_STATE>"FloorColour}"   { BEGIN STAGEVALUE_STATE;
                                strcpy(endroit, "Property:{FloorColour}");
                                #if defined(DEBUGGING_LEXER)
                                  fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                #endif
                              }

<STAGE_STATE>"StageColour}"   { BEGIN STAGEVALUE_STATE;
                                strcpy(endroit, "Property:{StageColour}");
                                #if defined(DEBUGGING_LEXER)
                                  fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                #endif
                              }

        /*  "-"?  denotes zero or one "-"          */
        /* I think it's exist negative colours !!! */
<STAGEVALUE_STATE>"-"?[0-9]*\n  { BEGIN EXITSTAGE_STATE;     /* Exit the Stage property */
                                yytext[yyleng - 1] = '\0'; /* Put out the "\n"        */
                                #if defined(DEBUGGING_LEXER)
                                  fprintf(FICHIER,"#STAGESTRING \\n : %s#\n",yytext);
                                #endif
                              }
```

```
<STAGEVALUE_STATE>"-"?[0-9]*"{"     { BEGIN STAGE_STATE;          /* Return to the Stage Property */
                                      yytext[yyleng - 1] = '\0'; /* Put out the "{"          */
                                     #if defined(DEBUGGING_LEXER)
                                       fprintf(FICHIER,"#STAGESTRING { : %s#\n",yytext);
                                     #endif
                                    }

          /* ************************************************************************ */
          /* ************************ StageObject Line ********************************* */
          /* ************************************************************************ */

<EXITSTAGE_STATE>"[Stage]{"         { BEGIN STAGE_STATE;  /* a second stage after an other one  */
                                      strcpy(endroit, "Header:[Stage]");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

<EXITSTAGE_STATE>"[StageObject]{"   { BEGIN STAGEOBJECT_STATE;
                                      strcpy(endroit, "Header:[StageObject]");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

          /* It's obligatory in a file for to have (That's the minimum) :
                - The [Visual Assistant 1.0] header
                - the [World] header
                - One [Stage] header
             Dus, it's just possible for to have the EOF after to have
             the tree previous headers. Into the bargain, the EOF it's
             just possible at the beginning of a new line! */
<EXITSTAGE_STATE><<EOF>>            { BEGIN ENDOFLEXER_STATE;
                                      strcpy(endroit, "End Of File");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#%s#\n",endroit);
```

```
                                      #endif
                                    }

<ENDOFLEXER_STATE>.*\n*             { strcpy(endroit, "IT'S IMPOSSIBLE ... AFTER EOF");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#%s#\n",endroit);
                                      #endif
                                    }

<STAGEOBJECT_STATE>"TimeStart}"     { BEGIN STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE;
                                      strcpy(endroit, "Property:{TimeStart}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

<STAGEOBJECT_STATE>"TimeStop}"      { BEGIN STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE;
                                      strcpy(endroit, "Property:{TimeStop}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

<STAGEOBJECT_STATE>"StorageOriginalBMP}"   {
                                      BEGIN STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE;
                                      strcpy(endroit, "Property:{StorageOriginalBMP}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }
<STAGEOBJECT_STATE>"StoragePatternedBMP}"  {
                                      BEGIN STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE;
                                      strcpy(endroit, "Property:{StoragePatternedBMP}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
```


Appendix – the lexer code

```
                                    }

<STAGEOBJECT_STATE>"StorageAngledBMP}"     {
                                      BEGIN STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE;
                                      strcpy(endroit, "Property:{StorageAngledBMP}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

<STAGEOBJECT_STATE>"Patterned}"    { BEGIN STAGEOBJECT_BOOLEAN_VALUE_STATE;
                                      strcpy(endroit, "Property:{Patterned}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

<STAGEOBJECT_STATE>"Center_3D}"    { BEGIN STAGEOBJECT_3D_COORDONATE_VALUE_STATE;
                                      strcpy(endroit, "Property:{Center_3D}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

<STAGEOBJECT_STATE>"Size_3D}"      { BEGIN STAGEOBJECT_3D_COORDONATE_VALUE_STATE;
                                      strcpy(endroit, "Property:{Size_3D}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

<STAGEOBJECT_STATE>"OffSetHoriz}" { BEGIN STAGEOBJECT_INTEGER_VALUE_STATE;
                                      strcpy(endroit, "Property:{OffSetHoriz}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

<STAGEOBJECT_STATE>"OffSetVert}"   { BEGIN STAGEOBJECT_INTEGER_VALUE_STATE;
                                      strcpy(endroit, "Property:{OffSetVert}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

<STAGEOBJECT_STATE>"BillBoard}"    { BEGIN STAGEOBJECT_BOOLEAN_VALUE_STATE;
                                      strcpy(endroit, "Property:{BillBoard}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

<STAGEOBJECT_STATE>"PreviousPositionFront}"  {
                                      BEGIN STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE;
                                      strcpy(endroit, "Property:{PreviousPositionFront}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

<STAGEOBJECT_STATE>"PreviousPositionTop}"    {
                                      BEGIN STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE;
                                      strcpy(endroit, "Property:{PreviousPositionTop}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                      #endif
                                    }

<STAGEOBJECT_STATE>"NewPositionFront}"       {
                                      BEGIN STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE;
                                      strcpy(endroit, "Property:{NewPositionFront}");
                                      #if defined(DEBUGGING_LEXER)
                                        fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
```


Appendix - the lexer code

```
                                #endif
                              }

<STAGEOBJECT_STATE>"NewPositionTop}"          {
                                BEGIN STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE;
                                strcpy(endroit, "Property:{NewPositionTop}");
                                #if defined(DEBUGGING_LEXER)
                                  fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                #endif
                              }

<STAGEOBJECT_STATE>"MergedRectangleFront}"    {
                                BEGIN STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE;
                                strcpy(endroit, "Property:{MergedRectangleFront}");
                                #if defined(DEBUGGING_LEXER)
                                  fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                #endif
                              }

<STAGEOBJECT_STATE>"MergedRectangleTop}"      {
                                BEGIN STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE;
                                strcpy(endroit, "Property:{MergedRectangleTop}");
                                #if defined(DEBUGGING_LEXER)
                                  fprintf(FICHIER,"#PROPERTY : %s#\n",endroit);
                                #endif
                              }


          /* The open { must to be on the same line that the rule to match !!! */
<STAGEOBJECT_BOOLEAN_VALUE_STATE>[Ff][Aa][Ll][Ss][Ee]"{"|[Tt][Rr][Uu][Ee]"{"      {
                    BEGIN STAGEOBJECT_STATE; /* Return to the StageObject Property */
                    yytext[yyleng - 1] = '\0'; /* Put out the "{"          */
                    #if defined(DEBUGGING_LEXER)
                      fprintf(FICHIER,"#STAGEOBJECT_BOOLEAN { : %s#\n",yytext);
                    #endif
                  }



<STAGEOBJECT_BOOLEAN_VALUE_STATE>[Ff][Aa][Ll][Ss][Ee]\n|[Tt][Rr][Uu][Ee]\n     {
                    BEGIN EXITSTAGE_STATE; /* Return to the StageObject Property */
                    yytext[yyleng - 1] = '\0'; /* Put out the "{"          */
                    #if defined(DEBUGGING_LEXER)
                      fprintf(FICHIER,"#STAGEOBJECT_BOOLEAN \\n : %s#\n",yytext);
                    #endif
                  }

<STAGEOBJECT_INTEGER_VALUE_STATE>"-"?[0-9]*"{"      {
                    BEGIN STAGEOBJECT_STATE; /* Return to the StageObject Property */
                    yytext[yyleng - 1] = '\0'; /* Put out the "{"          */
                    #if defined(DEBUGGING_LEXER)
                      fprintf(FICHIER,"#STAGEOBJECT_INTEGER { : %s#\n",yytext);
                    #endif
                  }

<STAGEOBJECT_INTEGER_VALUE_STATE>"-"?[0-9]*\n     {
                    BEGIN EXITSTAGE_STATE; /* Return to the StageObject Property */
                    yytext[yyleng - 1] = '\0'; /* Put out the "{"          */
                    #if defined(DEBUGGING_LEXER)
                      fprintf(FICHIER,"#STAGEOBJECT_INTEGER \\n : %s#\n",yytext);
                    #endif
                  }

<STAGEOBJECT_3D_COORDONATE_VALUE_STATE>"-"?[0-9]*";""-"?[0-9]*";""-"?[0-9]*"{"      {
                    BEGIN STAGEOBJECT_STATE; /* Return to the StageObject Property */
                    yytext[yyleng - 1] = '\0'; /* Put out the "{"          */
                    #if defined(DEBUGGING_LEXER)
                      fprintf(FICHIER,"#STAGEOBJECT_COORDONATE_3D { : %s#\n",yytext);
                    #endif
                  }

<STAGEOBJECT_3D_COORDONATE_VALUE_STATE>"-"?[0-9]*";""-"?[0-9]*";""-"?[0-9]*\n      {
                    BEGIN EXITSTAGE_STATE; /* Return to the StageObject Property */
                    yytext[yyleng - 1] = '\0'; /* Put out the "{"          */
                    #if defined(DEBUGGING_LEXER)
```

```
                    fprintf(FICHIER,"#STAGEOBJECT_COORDONATE_3D \\n : %s#\n",yytext);
                  #endif
                }

<STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE>"-"?[0-9]*";""-"?[0-9]*";""-"?[0-9]*";""-"?[0-9]*"{"    {
                  BEGIN STAGEOBJECT_STATE; /* Return to the StageObject Property */
                  yytext[yyleng - 1] = '\0'; /* Put out the "{"          */
                  #if defined(DEBUGGING_LEXER)
                    fprintf(FICHIER,"#STAGEOBJECT_COORDONATE_TRECT { : %s#\n",yytext);
                  #endif
                }

<STAGEOBJECT_TRECT_COORDONATE_VALUE_STATE>"-"?[0-9]*";""-"?[0-9]*";""-"?[0-9]*";""-"?[0-9]*\n    {
                  BEGIN EXITSTAGE_STATE; /* Return to the StageObject Property */
                  yytext[yyleng - 1] = '\0'; /* Put out the "{"          */
                  #if defined(DEBUGGING_LEXER)
                    fprintf(FICHIER,"#STAGEOBJECT_COORDONATE_TRECT \\n : %s#\n",yytext);
                  #endif
                }

%%

int main(int NumberOfArguments, char **Arguments)
{ char ReturnMessage[100];
  int code;

  /* I write all in an output file for to debug the lexer */
  #if defined(DEBUGGING_LEXER)
    if ((FICHIER = fopen("debug.txt","w")) == NULL)
     { perror("\n Can't create \"debug.txt\""); /* C Function */
       return -4;
     }
  #endif

  if (NumberOfArguments == 2)
   { if ((yyin = fopen(Arguments[1], "r")) == NULL)
      { strcpy(ReturnMessage, "\n");
        strcat(ReturnMessage, Arguments[1]);
        strcat(ReturnMessage, " : Can't open specified Saving File (return -1)\n");
        code = -1;
      }
     else
      { if (!yylex()) /* if (yylex() == 0)  */
         { strcpy(ReturnMessage,"\nSaving File OK (return 1)\n");
           code = 1;
         }
        else
         { strcpy(ReturnMessage,"\nSaving File KO (return -2)\n");
           code = -2;
         }
      }
   }
  else
   { strcpy(ReturnMessage,"\nBad number of arguments : put the name of the Saving File! (return -3)\n");
     code = -3;
   }

 if (code != 1) perror(ReturnMessage); /* if you have an error ! */

 #if defined(DEBUGGING_LEXER)
   fprintf(FICHIER,ReturnMessage);
   fclose(FICHIER);
 #endif

 return code; /* return the code for to see if you have an error */
}
```